7-1-1998

An Approach to remote process monitoring and control

John Dracos

AN APPROACH TO REMOTE PROCESS MONITORING AND CONTROL

by

John Dracos

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by:

Principal Advisor_____
Roy S. Czernikowski, Professor and Department Head

Committee Member_____
Muhammad E. Shaaban, Assistant Professor

Committee Member_____
Tony H. Chang, Professor

Department of Computer Engineering
College of Engineering
Rochester Institute of Technology
Rochester, New York
July 1998

RELEASE PERMISSION FORM

Rochester Institute of Technology

An Approach to Remote Process Monitoring and Control

I, John Dracos, hereby grant permission to any individual or organization to reproduce this thesis in whole or in part for non-commercial and non-profit purposes only.

_____

John Dracos

7/31/98

_____

Date

ABSTRACT

The purpose of this thesis is to present an approach to remote monitoring and operation of distributed real time process control systems. Conventional monitoring of process control systems currently requires a great deal of close supervision from trained personnel located on-site. In many cases, researchers, developers or maintenance personnel cannot be at every location where such a system is installed. Currently, a standardized architecture for remote access to such systems is not available. In addition, most of these systems are very expensive and under-utilized. Researchers would benefit by having access to different parts of a system concurrently.

The benefits of a layered architecture for remote process monitoring and control will be analyzed through the use of a demonstration system that was realized to examine the real time performance of the interconnection mechanisms between the process controller(s) and the system monitoring interfaces. Low level, real-time process control is achieved by using specialized networking schemes called fieldbusses to interconnect all control devices. In this system, fieldbus controllers will also assume the role of servers connected to the Internet, in order to make device information available to any local or remote clients. In the proposed architecture, remote clients are user interfaces, implemented as JAVA applets, which can be accessed with a web browser. The proposed system architecture allows for client interfaces to gain remote access to various types of fieldbusses transparently.

ACKNOWLEDGMENTS

The server software in this thesis uses sample source code supplied by *SST*. This software in its original form is used in conjunction with an *SST* scanner card to control a DeviceNet fieldbus network. I have extended and modified the software to provide Internet server capabilities to the application by implementing a protocol described in this thesis. I have also added minor diagnostic capabilities to the original application. The sample application source code, in its original form, was provided to demonstrate a way of using the Dynamic Link Libraries (DLL) that are available for the specific *SST* DeviceNet card.

Internet connectivity at the server side was achieved by use of *SocketWrench*, a freeware ActiveX control by *Catalyst Corporation*. *SocketWrench* encapsulates the functionality of socket connectivity and provides a uniform and portable interface of events and actions to the user.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ACTIVEX

Binary object that provides specific functionality and is intended for Internet applications. It is an extension of OLE for the Internet **64**

API

Applications Programming Interface. The way in which programs interface to a product or service. The programmer is aware of what services are offered and how services can be requested, but is not concerned with how the services are implemented. An API is often implemented as a library of functions **26**

APPLICATION GATEWAY

An internetworking unit which is responsible for conversions between the application layers of two different networks in order to provide a uniform applications environment **49**

BASEBAND

A network in which data is transmitted unmodulated as direct digital levels using the whole channel bandwidth. Since only a single channel is available, only one conversation can be supported at any given moment in time **16**

CAN

Controller Area Network. A fieldbus developed by Bosch GmbH. Originally intended for low cost communications among sensor-actuator devices found in automobiles. **14**

CHANNEL

A single transmission path through a medium to enable communication. A single medium may be capable of supporting multiple channels. **31**

CLIENT/SERVER

A model for structuring a distributed system. The system consists of two types of processes **50**

CONNECTIONLESS

A service which transmits data without expecting an acknowledgement from the receiver. Full addressing information is attached to each packet so that it can be routed independently. It is particularly appropriate when the channel error rate is relatively low. This type of service is also commonly termed a *datagram service* **17**

CONNECTION-ORIENTED

A service which first establishes a connection between sender and receiver before data is transmitted. After data transmission is complete, the connection is terminated..... **18**

CRC

Cyclic Redundancy Code. An error-detecting code that is based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. The sender and the receiver agree on a certain polynomial in advance. **16**

CSMA

Carrier Sense Multiple Access. An access control method which indicates that when a station has data to send, it first listens to the channel until it becomes idle. **16**

DP

Decentralized Peripherals. Description of the operation of state machines responsible for low level sensor/actuator communications using PROFIBUS. **21**

DROPLINE

Thin DeviceNet cable used as subnet carrier **16**

EIA RS-485

Electrical Industries Association RS-485. An electrical standard for balanced data communications, in which only two wires are required. Balanced transmission requires no common ground between a transmitter and a receiver. **16**

FDL

Fieldbus Data Link ..... *See* OSI

FMS

Fieldbus Message Specification. Soft real-time communications object model for PROFIBUS. **21**

HAMMING DISTANCE

The number of bit positions in which two codewords differ. Its significance is that if two codewords are a Hamming distance **d** apart, it will require **d** single-bit errors to convert one into the other. **25**

IDENT

Unique identification number given to every registered PROFIBUS device by the PROFIBUS Trade Organization **29**

ISO

International Standards Organization. An organization that mandates or recommends standards **14**

LAN

Local Area Network. A network covering a small geographic area, normally within a building or building complex **49**

LCA

LonWorks Component Architecture. Services that provide connectivity of network devices to data acquisition equipment. **30**

LNS

LonWorks Network Services. LonWorks Network API for developers of Windows-based user interfaces. **30**

LONTALK

The low-level communications protocol used by all devices on a LonWorks bus **30**

MAC

Medium Access Control. A sublayer of the Data Link layer which is used to determine which station can access a shared communications channel. **19**

MASTER-SLAVE

A model for structuring a centralized system (or combination of centralized and distributed system) in which master processes act as clients (requesting services from slave processes) and slave processes act as servers (providing services to master processes)... **16**

NRZ

Non-Return to Zero..... **16**

ODVA

Open DeviceNet Vendors Association. An organization responsible for maintaining and making DeviceNet standards available to users and developers **14**

OPC

OLE for Process Control. A server application that uses Microsoft's COM or DCOM to supply data to other applications..... **40**

OPEN STANDARD

A standard with specifications that are available to all interested users on equal terms and without restrictions..... **5**

OSI

Open Systems Interconnection. An ISO architectural reference model and a set of standards which describe how communication is achieved across systems from different vendors. The seven layers that are defined by this model are the physical, data link, network, transport, session, presentation, and application layers. **7**

PA

Process Application. Description of inter-process relationship modeling using PROFIBUS communications..... **21**

PEER-TO-PEER

A model for structuring a distributed system in which all peer processes are clones with the ability to act as clients (requesting services from other peer processes) and servers (providing services to other peer processes)..... **16**

PHY

PHYsical layer..... *See* OSI

PLUG-AND-PLAY

The ability of a system to automatically detect and configure a new device based on information that is provided by that device to the system controller..... **4**

PRODUCER-CONSUMER MODEL

An inter-process communication model according to which processes make data available to other processes by *producing* data and any interested processes *consume* that data.... **17**

PTO

PROFIBUS Trade Organization. A non-profit organization that provides user support, developer support, and product market exposure for PROFIBUS..... **29**

TOKEN PASSING

An access control mechanism that uses a special bit pattern called *token* to ensure that no two stations can transmit onto a shared medium at the same time..... **22**

TRUNKLINE

Thick DeviceNet cable used as main bus carrier..... **16**

TWISTED PAIR

A transmission medium consisting of a pair of copper conductors twisted around each other to improve noise immunity. It comes in two varieties. One is unshielded twisted pair (UTP) and the other is shielded twisted pair (STP) which includes extra protective shielding..... **16**

1 Introduction

For a long time, it has been a very tedious task for systems engineers to set up and configure a distributed monitoring and control system for an industrial application. Industrial applications usually require a large number of interconnected devices that provide inputs and outputs to one or more controllers in a configuration commonly called a control loop. These devices run an application that requires real-time responses. Such a control loop is usually called a *process*, or a recipe of how to realize the objectives of the system. These recipes contain ingredients that are the system variables and do not necessarily reside at a single location; they may be distributed among several different networked devices in the system.

Up until the late 1980's and for some through the early 1990's, process systems were set up as dedicated analog or digital connections of each device to a centralized control processor. Most industrial applications today that use this scheme incorporate an analog interface or a digital link for each device in the system. This interface receives and transmits the analog-encoded information from one device to another in a predetermined amount of time and implies, in turn, a certain scaling that is agreed upon in both devices [7, p.1]. This scaling is in fact the encoding of information into an analog signal. The large number of wires and cables proved to be a major problem in manufacturing and maintaining such systems. In addition, analog signals are not immune to electromagnetic noise and can be dangerous in intrinsically safe environments. Intrinsically safe environments are designated for high sensitivity to electronic switching (ex: possible ignition of volatile gasses due to electronic sparks). The realization of all the above led to the use of sophisticated digital industrial networking schemes for control and monitoring of processes and allowed for the migration to distributed processing.

This thesis is organized in a fashion that closely follows the architecture design of the proposed process control system. Section 2.1 describes the advantages of fieldbus networking, the lowest layer in the proposed system's architecture. Section 2.2 analyzes several considerations and issues that relate to the use of fieldbus networking in real-time control systems. Section 2.3 presents an analysis of specific requirements for implementing distributed control systems. This section presents requirements that apply to both the fieldbus communications layer and the client-server layer of the proposed architecture. Finally, section 2.4 presents previous work in the field of teleoperation and telemonitoring of control systems over the Internet. A great deal of research has been taking place in teleoperation and telemonitoring of robotic systems over the past few years. The need for universal telerobotics has sprung from the limited availability of robotic systems and the high maintenance cost of those systems in manufacturing and research environments.

Chapters 3,4 and 5 present detailed technical information for three popular fieldbus networking schemes. This information provides a useful reference about emerging fieldbus technologies, and demonstrates that the choice of a specific fieldbus network does not impact the architecture of the proposed monitoring and control system. The three fieldbusses are then briefly compared in chapter 6 and possible architectures and technologies based on those fieldbusses are presented in chapter 7.

Section 8.1 deals with general client requirements in remote monitoring and control systems. Section 8.2 presents the proposed system requirements, considerations, and architecture. An analysis of the advantages and disadvantages of this architecture provide a good insight into the feasibility of such a solution and its value in real world applications. Sections 8.3 and 8.4 proceed with an analysis of the client-server communication interface and protocol.

Chapters 9 and 10 conclude this thesis with a presentation of experimental results, problems that were encountered during the design and development of the proposed system, suggestions for future improvements and research, and a list of required developer resources.

User manuals for the client and server modules of the proposed system are presented in appendices at the end of this paper. A CD-ROM with software source, documentation and software installation is also included.

2 Background – Previous work

2.1 Fieldbus networks

"Fieldbus" and "Device bus" are two of the most popular terms in industrial communications today. What they represent is a large and still growing number of digital networks intended mainly for manufacturing applications such as semiconductor and storage media manufacturing. Fieldbus networks allow low-level devices such as real-time data acquisition equipment, sensors and actuators to be connected via a single medium and to communicate at high data rates, following a predefined signaling protocol [4, p.93].

A single network cable replaces the number amount of wires and cables which connect each pair of devices in a conventional control system. This network cable connects all devices in a bussed configuration. It distributes the digital network signals and, depending on the fieldbus, a DC power supply that can be used to provide power to network devices. Each device has usually enough processing power to perform functions required to adhere to the communications protocol and also process the actual data it received from the network. This processing power is usually obtained at a minimal cost and the distribution of control logic that makes the devices 'smart' yields increased system robustness, modularity and fault-tolerance [4, p.93].

In general, industrial networking offers lower installation costs and maintenance over the lifetime of a system. Whenever maintenance or troubleshooting is required, malfunctioning components can be replaced possibly without disturbing the network (*Hot Insertion/ Extraction* of devices). The same principle applies to upgrading field devices. Enhanced diagnostic capabilities that are available with industrial networking schemes enable predictive maintenance and provide complete information about the status of a device or the entire system. Fieldbus networking schemes also offer plug-and-play capabilities because device models can be developed for various standard

sensors/actuators that are used in a system. Then, devices with more advanced capabilities can extend the basic functionality of the original-standard models; this can be considered as firmware inheritance, which reduces development costs and provides basic compatibility between devices from different vendors. This property of a device, called *interchangeability*, enables the substitution of such a device by a comparable one by assuring identical functionality with the original device. Furthermore, interchangeability increases the lifetime of a system because the system does not become obsolete when one of its components becomes obsolete. Another property, called *interoperability*, enables a fieldbus device to function correctly and predictably in a network of devices with similar capabilities. Fieldbus equipment manufacturers and suppliers, especially in the semiconductor industry, deemed these two properties necessary to allow for integration of flexible and easy to maintain systems [23, pp. 8,16]. This is possible because most fieldbus networks are usually supported by widely accepted, *open standards*. Open standards give users and designers the ability to choose equipment from a number of standard-compliant vendors.

To summarize, there are many advantages of fieldbus networking [23, pp.16-17] but the majority of them belong in one or more of the following categories:

- Reliability
- advanced diagnostics
- ease of installation
- standardized physical layer
- reduced cost
- expandability and size
- flexibility
- process improvement
- sophisticated network architecture
- fault tolerance
- real-time performance

2.2 Real-time control using Fieldbusses

There is currently a great variety of fieldbus networking schemes installed and used in many manufacturing or research applications. These fieldbus networks apply specific communication protocols to distinguish and prioritize different message types that may be transmitted across the network. Some of these fieldbus networks go as far as implementing structures – frameworks for mapping entire distributed control systems to network variables or parameters. Others, implement object oriented hierarchies that allow the user or developer to map distributed control system attributes to network objects. At the lowest level of protocol implementation, messages in a fieldbus network used for real-time control can be classified into three categories [4, pp.94-95]:

- *hard-deadline periodic messages* are responsible for the transfer of most of the distributed system control parameters at a constant rate. Loss or duplication of these messages can be detrimental to the operation of the system.
- *hard-deadline sporadic messages* are used for transfer of equally critical information that could be the report of an unexpected process fault or warning.
- *non-real-time aperiodic messages* are used for transfer of less critical information, such as system configuration, or for health monitoring of network devices.

To account for the various delivery deadlines, some fieldbusses prioritize messages and guarantee delivery within a certain amount of time from the moment the request is transmitted to the moment when a response is obtained. In case of collisions between two or more devices trying to gain access to a shared medium (fieldbus), techniques such as no-penalty arbitration are used to resolve these collisions on the fly. No-penalty arbitration, which is used by DeviceNet, resolves collisions by defining a *dominant* and a *recessive* bit level on the bus. Dominant bit levels always win over recessive bit levels. Every device checks the bus bit level immediately after transmitting each bit. Whichever

device transmits a dominant bit over other devices' recessive bits is allowed to continue transmission, while the other devices postpone transmission [9].

Other fieldbusses simply rely on high data rates to probabilistically resolve collisions. Just as in the popular Ethernet-based LANs, these fieldbusses allow devices on the bus to transmit at will; when a device encounters a collision with one or more other devices trying to transmit, it postpones transmission for a random period of time and then retries [16, pp.252-253].

Even though the majority of well known fieldbusses follow the predefined OSI model for layered architecture of networks [16, pp. 28-44], not all of these fieldbusses implement and distinguish between all seven layers in the OSI model. For efficiency purposes, some of the functionality of the middle layers (session, transport and network layers) is assigned to either the data link or the application layer. For example, no routing is necessary in a bus communication scheme because any message can get from the source to a destination in one hop. That is why these busses are commonly referred to as *multi-drop busses*. Thus, in order for real-time deadlines to be satisfied, at least two layers in these fieldbus architectures have to be involved. First, the transport layer (or any other layer that is responsible for managing transport functions) must prioritize messages [4, pp.94-95] and manage multiple messaging connections by tracking transmission latencies and possibly submitting timeout events to higher layers. The data link layer is then responsible for arbitration, collision resolution or retries.

An implied consideration with devices that implement interfacing to popular fieldbusses is how the device firmware responds to the requirements of other real-time tasks being handled by the device. For example, a complex sensing device that needs to relay the sensory information to interested parties over the fieldbus, must also be able to collect this information in real-time from its environment. In that case, how well does the

firmware handle very high-speed network requests and, at the same time, handle real-time deadlines required by its sensors? The following classes of scheduling algorithms are defined for real-time kernels [3, pp. 56-57]:

- ***Static table-driven scheduling***: An analysis of tasks and scheduling requirements is done before execution. The resulting schedule is stored in a table that can be referenced to at run-time.
- ***Static priority-driven preemptive scheduling***: Analysis of tasks and scheduling is done as in the static table-driven approach but at run-time, tasks execute "highest priority first"
- ***Dynamic planning-based scheduling*:** Task execution feasibility is dynamically examined at run-time. If found feasible, tasks are guaranteed to meet their real-time deadlines.
- ***Dynamic best effort scheduling*:** System tries to schedule tasks so that task deadlines will be met, but provides no guarantees for the tasks actually meeting those deadlines.

In the above classification, static approaches usually require less processing at run-time but are less flexible. To increase scheduling flexibility for aperiodic tasks and reduce scheduling time for periodic tasks, a hybrid of static and dynamic scheduling methods can be used. Various performance metrics exist for fine-tuning these scheduling algorithms [3, p. 57]. Real-time systems usually encounter tasks with various:

- computation requirements
- resource requirements
- importance levels (priorities)
- precedence relationships

- communication requirements
- timing constraints

For most low-level sensor/actuator devices used in complex distributed control – process systems, it is usually more efficient and cost effective to use small proprietary kernels as the core of the sensor/actuator software. The kernels can be easily optimized [3, pp.61-62] for:

- fast context switching
- small software and physical size
- timely response to external interrupts (from environment sensor components)
- good memory management
- on-the-fly data logging of real-time information

The time-critical tasks that such kernels deal with require the use of a real-time clock (usually provided by a precise clock oscillator), priority scheduling to differentiate between time critical periodic tasks and deadline-flexible periodic or aperiodic tasks, special alarms and timeouts (especially for communication timeout events) and real-time queuing algorithms.

2.3 Communication principles in distributed control systems

Networking of industrial control systems, is based on the fact that these systems have evolved a great deal in the last decade. The architecture and functionality of industrial control systems has changed to allow for intelligent devices to perform distributed I/O functions and remote monitoring.

From the perspective of system architecture as it relates to the actual implementation of a control process, it becomes very natural to assume that a great deal of device-level I/O data can be processed locally at the device and the result that is of importance to the control process can be derived and communicated. In other words, optimal distributed control system configurations would allow much of the I/O formatting and processing work to be done at the device. From the perspective of the inter-network architecture this would not necessarily simplify the protocol that is used but would reduce network traffic and increase performance.

There are several general requirements for communications in distributed systems and most importantly for inter-networking of process systems [2, pp. 75-77]:

- **Transparency**: process systems usually consist of large number of devices, each one using one or more processors to implement communications tasks, gather or distribute sensor information and control other devices in the system. First, these devices may not be part of the same physical network and thus will not use the same method of communication to transfer information between them. Second, if there is more than one path of communication between two devices, the most efficient path must be followed in order to maintain high performance. A transparent distributed control system must provide a consistent mechanism for one process to communicate with other processes in the system by hiding the complexity of inter-networking. In other words, every process should be able to view any other process in the system as if it were connected to the same physical network.
- **Remote Procedure Call (RPC)**: at the lowest level, distributed systems implement communication of device inputs and outputs from one device to the other over a deterministic network. However, to implement a distributed control system, devices have to request data and respond to requests from other devices. This request-

response mechanism obeys the client-server model, where the requestor of information is the client and the information provider is the server. Usually, every device in a distributed system assumes both the client and the server roles at different times. RPC is essentially a set of primitives available as libraries of procedure calls that enable message transfers and message arrival indications.

- **Group Communication**: it is usually the case that a large number of processes in a distributed control system need to interact closely and repeatedly either during normal operation (ex. closed loop control) or initial configuration (ex. initialize device I/O). Also, in some process applications where fault-tolerance or distributed-shared memory is required, data can be broadcast to multiple receivers. In a distributed system where selective group communications (multicasting) or system wide communications (broadcasting) are available as communications primitives the performance is always noticeably higher than that of a system where group communications are achieved by n single point-to-point messages. The only drawback of group communications available as a primitive is that group participation tables must be maintained and kept up to date to ensure correct operation.

- **Security**: in most distributed systems, data encryption is implemented on top of the communications protocols and is, therefore, applied to any message that is sent using those protocols. In other words, a process or device cannot selectively bypass the security algorithms in order to increase performance and insure determinism. Systems that do provide selective deployment of security algorithms only during transmission through unsecured networks are necessary for real-time operation.

- **Network Management**: it has already been mentioned that system maintenance accounts for a large percentage of a system's lifetime. Manual configuration and user interventions are costly and time consuming. Also, maintenance on a single device can disrupt the operation of many or all the devices in the system. A distributed system that accounts for modularity, redundancy and minimal user intervention is considered superior to one that doesn't.

- **Wide-Area Networking**: distributed systems are usually structured in logical clusters so that, devices that communicate closely are connected to the same network. This improves performance and minimizes unnecessary delays caused by bridging over different networks frequently.

2.4 Teleoperation and Telemonitoring

Today, distributed control systems appear in many research facilities and manufacturing plants. It is usually the case that those systems are arranged in a hierarchical order. For example, a plant may consist of multiple clusters of various manufacturing capacities, and those clusters may consist of subsystems of various kinds of equipment, and so on. In any case, manufacturing plants and research facilities have many common characteristics. Both use costly, and usually, sophisticated equipment that is greatly valued for its 'up-time' In both areas, manufacturing engineers, researchers and developers constantly compete for system time even if only a small part of the system is needed or necessary system time by the developer is very short. Because of the cost of operation and, more importantly, maintenance of these systems, access is strictly enforced to a few people and great security measures are taken to ensure no unauthorized use of these systems not only for privacy, but also for safety.

Zhen, Lewis and Tan [6, pp.1400-1401] list a few of the setbacks that these access restrictions create. Their description is specific to robotic resources but they apply to many manufacturing and research facilities. First, the limited availability of costly latest technology equipment does not allow researchers to access these systems remotely. Maintenance of a typical system accounts for 80-90% of the system's lifetime; considering this, it is more likely to find maintenance crews in a lab than developers and researchers.

Even if researchers manage to get system time, limited or no standardized experimental facilities force them to spend more time searching for the right equipment and wondering about the validity of experimental results. In addition, for many experiments, measurements over time are far more valuable than initial data. Thus, reliable data logging and monitoring equipment are necessary. In situations where all the necessary equipment is actually located in a well organized section of a plant or a lab, it is still very hard to become familiar with a system because of equipment differences from supplier to supplier, interface differences, and non-standardized system configurations.

For many manufacturing applications, it is usually the case that each distributed system is operating in locations that are hazardous or not easily accessible to system operators. In the case of semiconductor manufacturing, these systems operate inside cleanrooms, locations with very high maintenance costs. Safety is always an important consideration, and maintenance often accounts for 90% of the lifetime of a system, so the need for remote monitoring and control of such systems is obvious.

Another reason for remote operation is the degree of integration of smaller distributed systems into large manufacturing processes. The monitoring of information from smaller sub-systems is equally important to the complete picture of the distributed process that is running in the system. For example, a mini disc manufacturing system with 6 tools (can produce 6 mini discs per run) can be monitored as a whole, or as 6 separate and independent mini disc sub-systems.

Finally, when multiple manufacturing processes are running in the same plant, remote operation enables plant engineers to monitor multiple processes from one or more locations around the plant or around the world. For the same reasons, equipment suppliers and applications engineers can monitor a system, perform remote data logging and troubleshooting.

3 DeviceNet$_{TM}$

3.1 Where it originated

This industrial fieldbus was first introduced by Allen-Bradley in March of 1994. Currently, Open DeviceNet Vendors Association (ODVA) is the organization that supports this very widely used bus and provides developers and users with specification documents [9]. Since the bus architecture is an open architecture, any one who is interested can use the technology and protocol specifications to provide a DeviceNet interface to their device/sensor. The Controller Area Network (CAN) protocol, on which DeviceNet is based, was developed by Bosch GmbH in Stuttgart, Germany [24]. CAN was originally intended to enable communications among devices such as anti-lock brake controllers or engine control units on a single bus. The high reliability and low cost of this device-level bus made it very successful in the automotive world.

Currently, DeviceNet is used in mainly manufacturing applications that include low level sensor-actuator devices for process control and feedback. The key characteristics of this standard are low cost, low maintenance and high interchangeability between products of different vendors.

The current standards for DeviceNet are ISO 11898 [25] and ISO 11519 [26]. There are currently six controller and transmitter IC vendors and hundreds of products that support DeviceNet.

3.2 Network architecture

The bus architecture of DeviceNet is based on the use of two separate types of cabling, trunk and drop cables, as seen in Figure 1. Both cables consist of two signal wires, two network power wires and a shield. Trunk cables are thicker than drop cables and are used as the main bus carrier. At various points in the network, drop cables are connected through taps to distribute data and power to physical subnets.

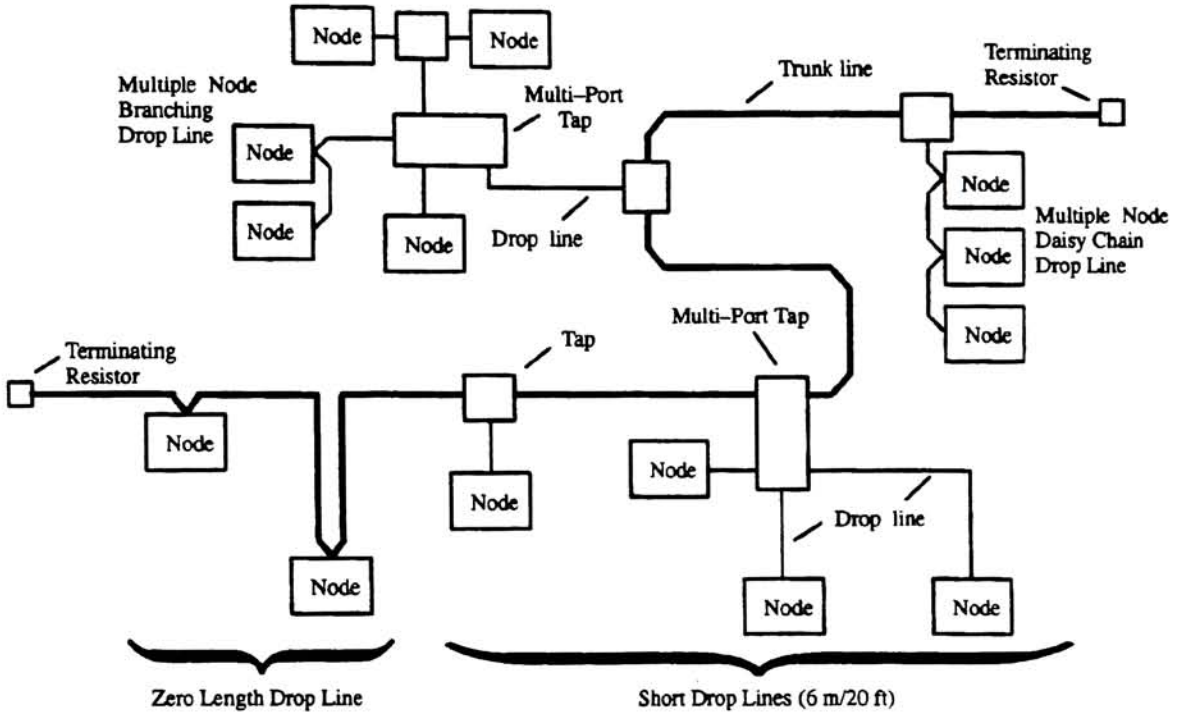

Figure 1: DeviceNet Media Topology

The Physical/Media specific characteristics of DeviceNet can be summarized as follows:

Signaling	CAN (Bosch GmbH)
Modulation	Baseband
Encoding	NRZ with bit stuffing
Media Coupling	DC coupled differential Tx/Rx
Physical media	Twisted pair for signal and power
Isolation	500 Volt (optional opto-isolators on node side of transceiver)
Topology	Trunkline - Dropline configuration
Max. number of nodes	64
Data rates	125 KBaud, 250 KBaud, 500 KBaud
Max. frame length	8 bytes per segment, maximum of 64 segments
Communication scheme	Master-slave, peer-to-peer.
Max. bus distance	500 m at 125 KBaud, 250 m at 250 KBaud, 100 m at 500 KBaud
Max. bus line voltage	-25 to +18 Volts referenced to transceiver ground
Differential Output level	2.0 Volts p-p (nominal), 1.5 Volts p-p (minimum)
Network power supported	Yes
Hot insertion/removal of nodes	Yes
Arbitration method	Carrier Sense Multiple Access / Collision Resolution (CSMA/CR)
Error checking	CRC

Table 1: Summary of DeviceNet characteristics

Notice that the physical layer signaling is similar to but not compatible with the EIA RS-485 standard [28].

3.3 Communications protocol

The DeviceNet communication stack is based on the ISO-OSI (Open Systems Interconnection) 7-layer model that describes the cumulative protocol structure for information transfer between two applications over a communications medium. DeviceNet selectively implements some of the OSI layers and not all 7 in order to maximize performance. Figure 2 shows that part of layers 1 and 2 are defined by the CAN protocol. Several standard definitions exist for the interface to the application layer.

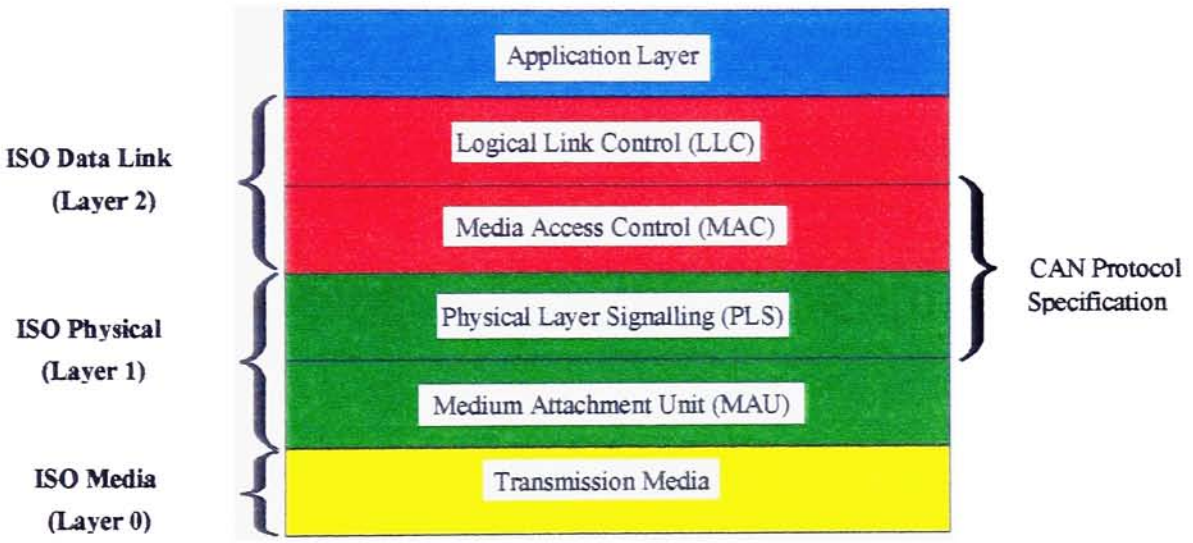

Figure 2: Mapping of DeviceNet to ISO-OSI model

One of the key characteristics of this communications protocol is that it is entirely defined as an object-oriented model based on the *producer-consumer* model. Since DeviceNet is a broadcast-based network, communications are carried out by one node producing data and one or more nodes consuming that data. The object-oriented structure of the protocol stack at each node allows for a modular development, testing and maintenance. Figure 3 shows the object model for the DeviceNet communications stack. DeviceNet supports connection-oriented and connectionless communication. In

connection-oriented communications, a predefined master/slave connection set or a peer-to-peer scheme can be used.

When individual objects are interrogated, *Explicit messages* are used. Once an *explicit messaging connection* is established, messages which contain a node ID, a message type, a class number, an object number and an instance number are used to enable communication between two objects in different devices on the network. The message router object on each device distributes requests to selected objects in the device. These messages are mainly used for housekeeping of the network, device configuration and diagnostics, or allocation/de-allocation of device connections and are of low priority.

For high speed I/O between devices on the bus, other types of messages are defined. These messages have the highest priority on the network as opposed to explicit messages. I/O messages can be produced as a result of either a single or broadcast request from another device (polled I/O), a change of state in one of the I/O channels in the device itself (change-of-state I/O), or a periodic I/O update event (cyclic I/O).

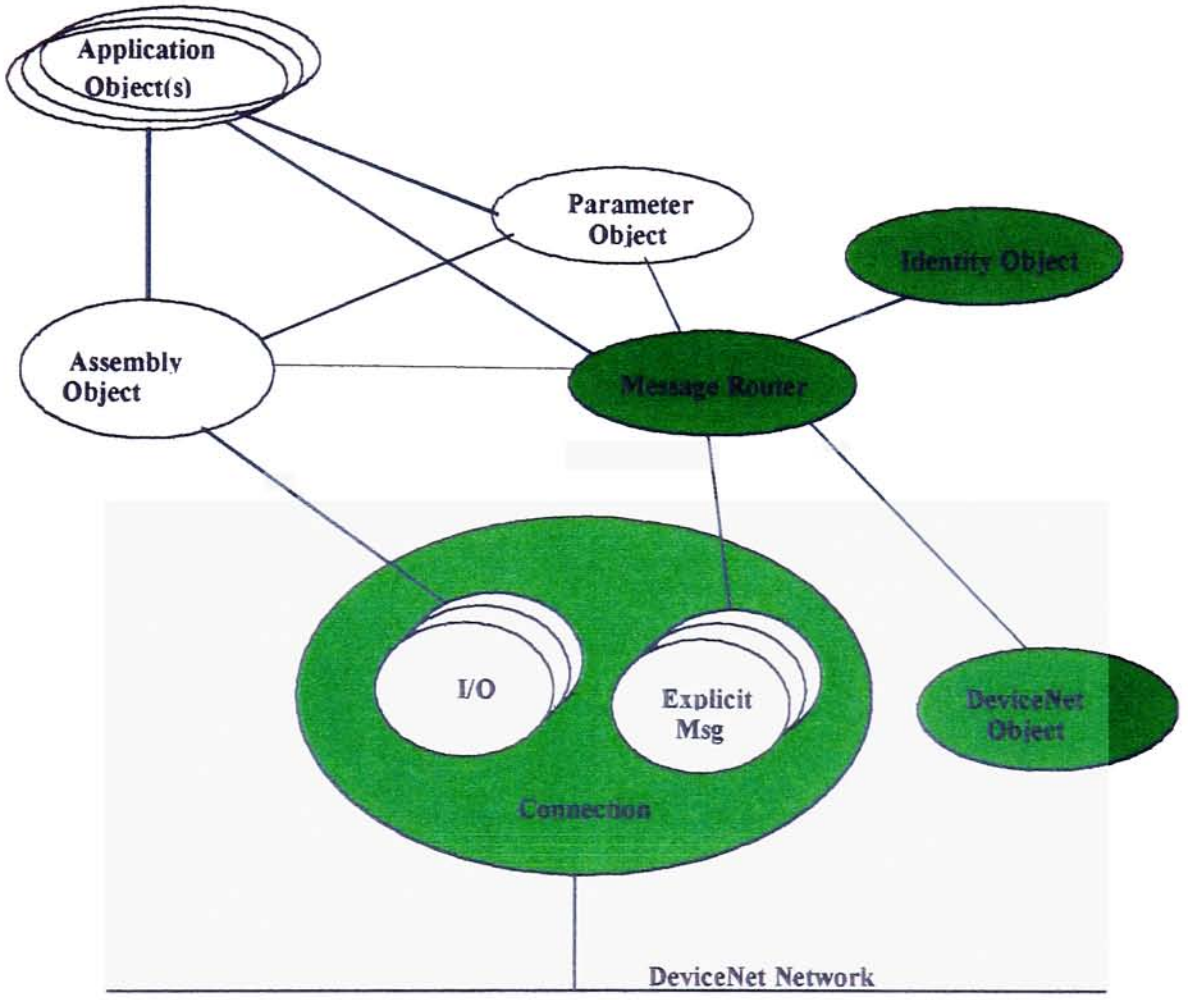

Figure 3: DeviceNet Object Model

In Figure 3 the following classes are mandatory for any device: Connection Class, DeviceNet Class, Message Router Class, Identity Class. The Identity object contains information such as vendor ID, device type, product code, revision number, status, serial number, product name and state. The DeviceNet object includes information about the MAC ID (node address), baud rate, and error counters. The Connection class instances describe and manage all the connections that the device is capable of allocating to the network. As previously mentioned, the explicit messages are distributed to the appropriate

device object through the Message router object. Finally, I/O data from various connections can be grouped together in logical assemblies that are processed by the application [9].

3.4 Documentation and Support

To prove conformance to the DeviceNet standards, every device that supports the protocol can be tested in various network configurations and loads by the supporting organization, ODVA. The DeviceNet standards also define a format for the documentation of the features of each device as they relate to communications.

Interoperability and interchangeability are the two main goals of this networking standard. To support interoperability, all devices on the DeviceNet communication link must support identical communication data and a common identity. To support interchangeability, devices of the same type must exhibit the same behavior, produce and/or consume the same basic set of I/O data, and contain the same basic set of configurable attributes. The formal definition of this information is known as a device profile. A device profile contains an object model, the I/O data format, and configuration data and the public interface(s) to that data for the device.

4 PROFIBUS$_{TM}$

4.1 Where it originated

Almost three and a half years after the German Federal Minister for Research and Technology initiated a collaborated project called *'Field bus'* in 1987, a German standard open field bus system was introduced. It was the joint collaboration of 13 companies and 5 institutes to create a high speed, reliable communications standard for use in process control applications. PROFIBUS (**PRO**cess **FI**eld **BUS**) which is based on the German standard DIN19245 [11] and the European Fieldbus standard EN50170 actually consists of three different protocols:

- PROFIBUS-FMS (**F**ieldbus **M**essage **S**pecification) which describes the communications object model from the point of view of the communication partner (server behavior) and is in general not used for hard real-time operations. This part of the protocol is the actual specification for application processes, operating systems and communication drivers. The PROFIBUS communication model allows for distribution of application processes that are part of a total process over the network via communication relationships.
- PROFIBUS-DP (**D**ecentralized **P**eripherals) which describes the cyclic exchange of data between powerful master devices, such as programmable logic controllers (PLCs) and smart sensor-actuator devices (decentralized peripherals - slaves) via a very high-speed reliable and deterministic serial channel.
- PROFIBUS-PA (**P**rocess **A**pplication) which describes the inter-process relationships in process control applications.

The most commonly used protocol is PROFIBUS-DP. It allows for a master-slave configuration with multiple masters. Communications between masters on the same bus is

achieved by the use of a token passing scheme and communication between masters and slaves is based on a request - response relationship. Before communications begin, one master allocates a connection to a set of slaves and configures them for data exchange. Once the data exchange is initiated, only the master device, which has allocated a specific slave device, can modify the slave device's data. Until the allocating master releases the slave device, any other masters on the bus can only read the slave device's data or diagnostic information.

4.2 Network architecture

The bus architecture of PROFIBUS is based on the use of two different communications media depending upon the performance and network span requirements of the application. For high speed, long haul, fieldbusses a fiber optic medium is used to allow for bus lengths of up to 24km. For smaller networks or local subnet structures shorter than 1.2km, a twisted pair medium is used. In practice, it is usually the case that the fieldbus master(s) (Programmable Logic Controllers, Numerical Controllers, Man-Machine Interfaces) control the network from a central location via a fiber optic medium. Slave devices are then organized in small logical subnets that are connected to a twisted pair medium and a local (to each subnet) fiber optic to twisted pair converter. Another common topology involves including repeaters between stations or subnets to extend the network's reach. Figure 4 and Figure 5 show some common topologies.

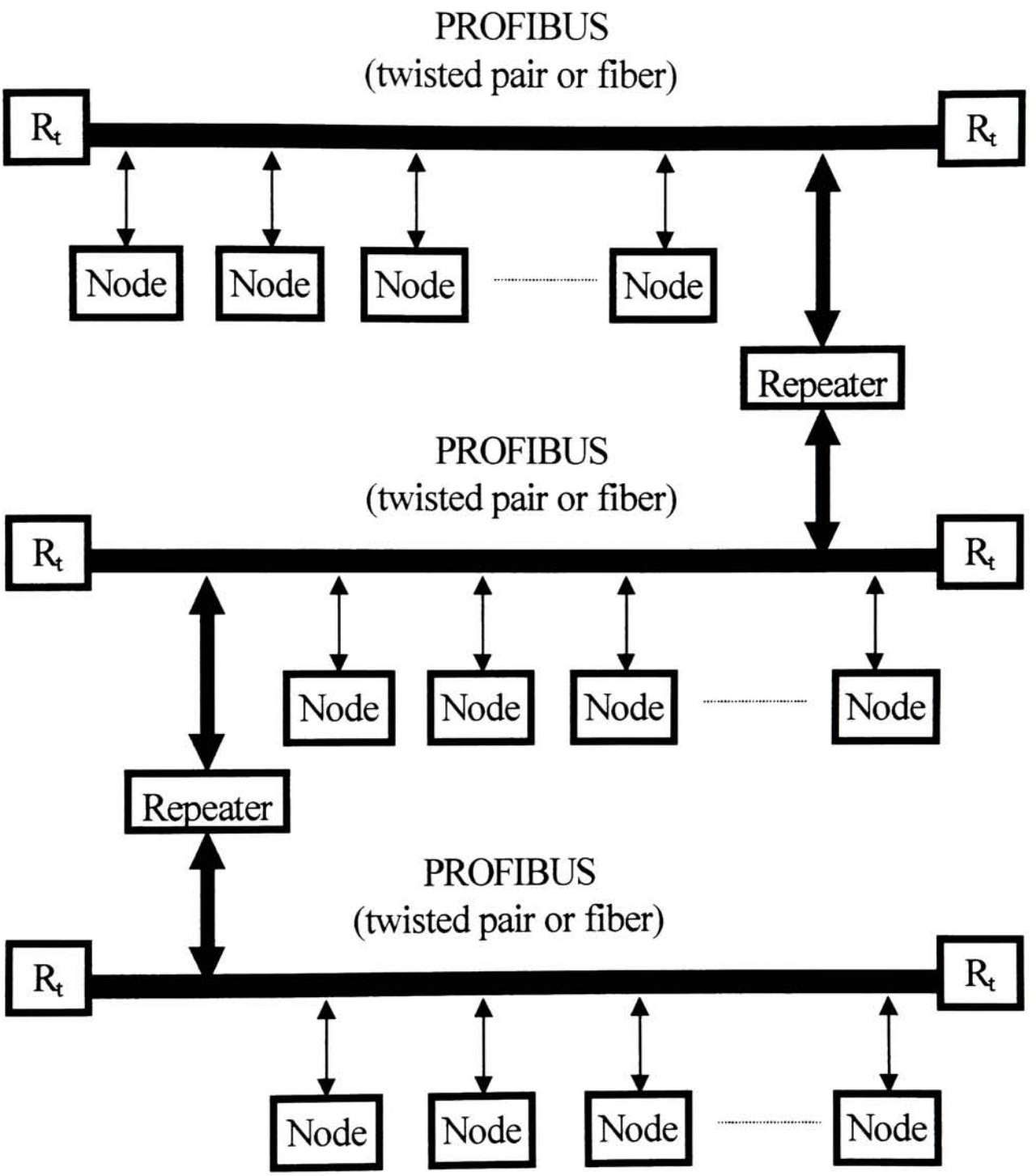

Figure 4: PROFIBUS Media Topologies: linear bus

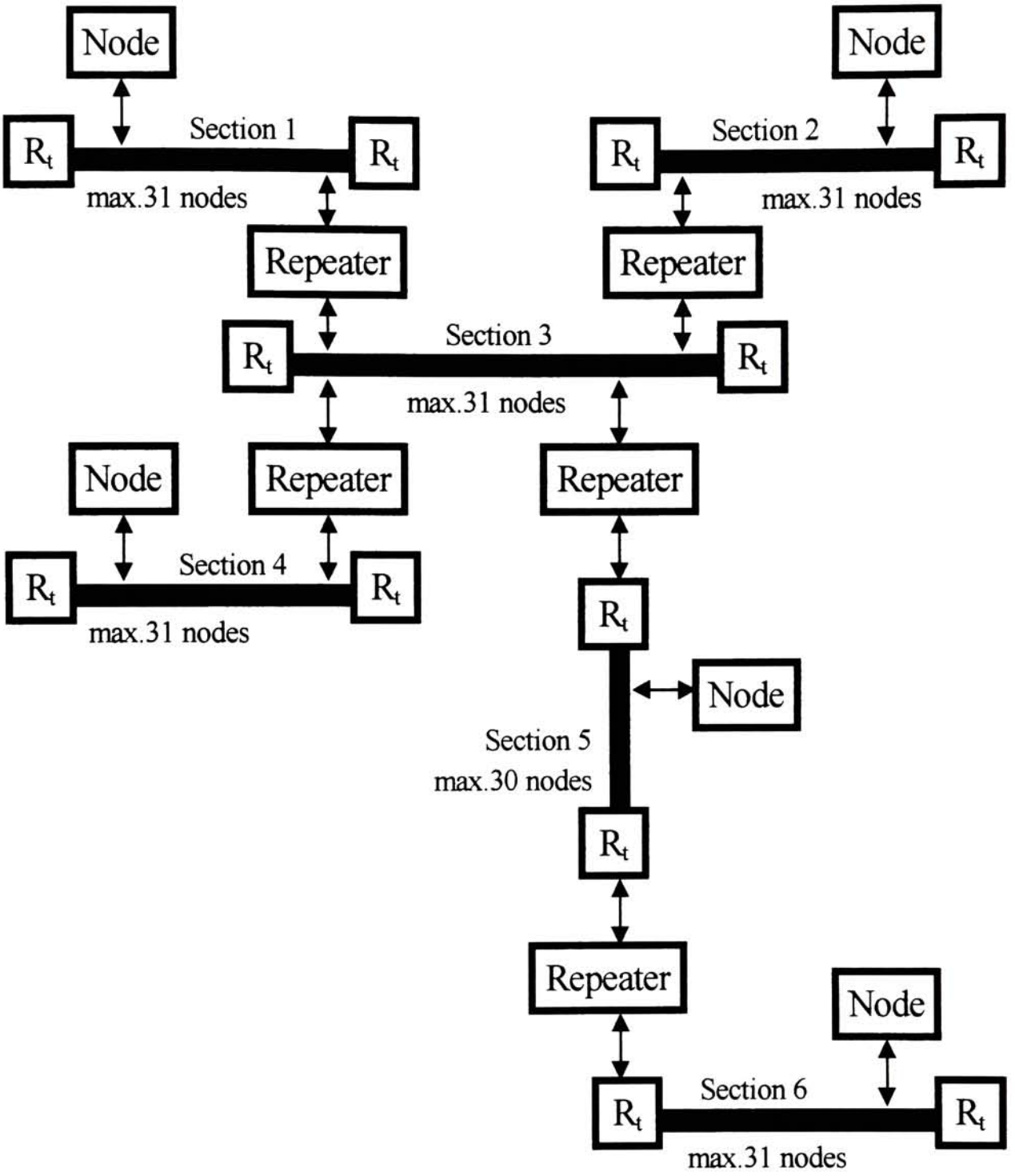

Figure 5: PROFIBUS Media Topologies: Tree

The Physical/Media specific characteristics of PROFIBUS can be summarized as follows:

Signaling	EIA RS-485 compliant
Modulation	Baseband
Encoding/Synchronization	Slip protected synchronization (no bit stuffing)
Media Coupling	DC coupled differential Tx/Rx
Physical media	Twisted pair, fiber
Isolation	500 Volt (optional opto-isolators on node side of transceiver)
Topology	Linear terminated bus, star, and ring.
Max. number of nodes	127 (preferably not more than 32 masters)
Data rates	DP: 9.6Kbaud - 12.0MBaud, PA: 9.6Kbaud - 31.25Kbaud
Max. frame length	244 bytes (no segmentation)
Communication scheme	Master-slave, peer-to-peer.
Max. bus distance	1200m for < 93.75Kbaud, 600m for < 187.5Kbaud, 200m for < 500KBaud, up to 24Km with fiber.
Max. bus line voltage	-25 to +18 V referenced to transceiver ground
Differential Output level	2.0 Volts p-p (nominal), 1.5 Volts p-p (minimum)
Network power supported	No
Hot insertion/removal of nodes	Yes
Arbitration method	Token passing
Error checking	Hamming Distance (HD) = 4 CRC

Table 2: Summary of PROFIBUS characteristics

Notice that the physical layer signaling is based on the EIA RS-485 standard.

4.3 Communications protocol

The PROFIBUS communication stack is based on the ISO-OSI 7-layer model that describes the cumulative protocol structure for information transfer between two applications over a communications medium. Much like DeviceNet, PROFIBUS selectively implements some of the OSI layers and not all 7 in order to maximize performance. **Figure 6** shows the lower level layers: Physical layer (PHY) and Fieldbus Data Link layer (FDL) that are a mandatory part of a PROFIBUS-DP network interface implementation. The physical layer (PHY) specification defines all the necessary characteristics of bus interfaces and transceiver logic. The data link layer (FDL) comprises the network access state machine. In PROFIBUS-DP the Application layer 7 is not used for simplicity and performance reasons. Instead, a user interface is defined in place of the (FMS), and a Direct Data Link Mapper (DDLM) is defined in place of the Lower Layer Interface layer (LLI). The user interface is responsible for accessing functions in the DDLM. In essence, DDLM is an application-programming interface (API) to actual PROFIBUS network functions. The DDLM consists of predefined DP communications functions that map to network state changes at the FDL.

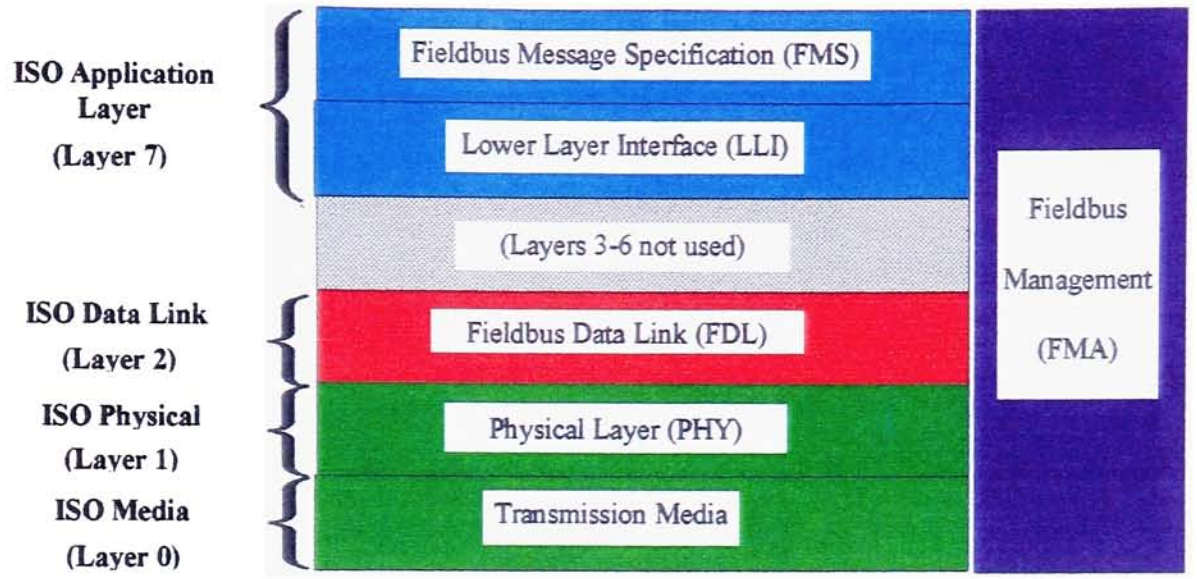

Figure 6: Mapping of PROFIBUS to ISO-OSI model

The model for PROFIBUS-FMS is object-oriented but communication is not deterministic and therefore it cannot be used for real-time process control. On the other hand, PROFIBUS-DP can meet hard real-time deadlines placed by various processes but there is no current model for communications that is object oriented. On a DP network interface communications are carried out via DDLM function calls and according to the network access state machine. The following flowchart shows the basic functionality of the network state machine for a slave device:

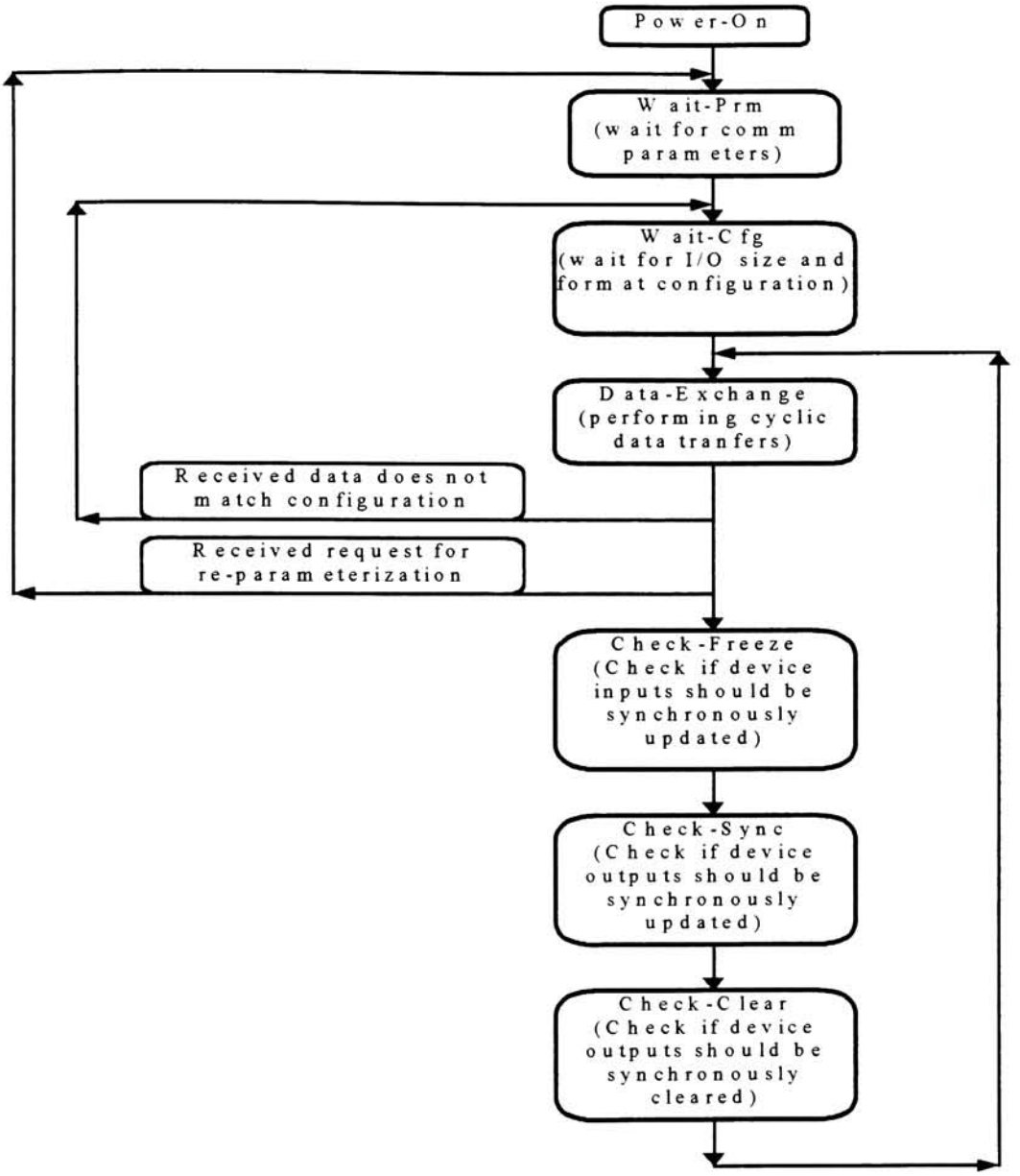

Figure 7: PROFIBUS-DP slave device network state machine

4.4 Documentation and Support

To prove conformance to the PROFIBUS standards, every device that supports the protocol can be tested in various network configurations and loads by the supporting organization, PROFIBUS Trade Organization (PTO). In a similar manner to DeviceNet specifications, the PROFIBUS standards also define a format for the documentation of the features of each device as they relate to communications. New devices are given an identification number (IDENT) by the PTO to distinguish them from other devices. The IDENT along with other communication characteristics of a device (maximum operating baud rate, size of I/O exchange, timing characteristics, etc.) must be documented in a database file of standard format (GSD file). In addition, some master devices can import GSD files and use the information to configure the network automatically (no need for manual entry of devices).

5 LonWorks®

5.1 Where it originated

LonWorks was developed by Echelon in Palo Alto, California and was introduced in March of 1991. It is also an open architecture whose protocol specifications are readily available to developers. In addition to protocol specifications for network device development, extended network services (LonWorks Network Services - LNS) are available for developers of Windows-based user interfaces.

Currently, LonWorks is mainly used in manufacturing applications that include low level intelligent sensor-actuator devices for process control and feedback. The key characteristics of this standard are low cost and maintenance and interchangeability between different vendors of equivalent products. The LonWorks family is an integrated technology for distributed process control and consists of the following parts:

- LNS (**L**onWorks **N**etwork **S**ervices): A scaleable architecture that adopts the client-server model to create a network framework for all services.
- LCA (**L**onWorks **C**omponent **A**rchitecture): Implements OLE (**O**bject **L**inking and **E**mbedding) to connect network devices to man-machine interfaces and data acquisition equipment.
- LonTalk: This is the low-level communications protocol used by all devices on a LonWorks bus. It defines the operation of a distributed logic and control system based on data from manufacturing and process equipment.

The current standard for LonWorks is ASHRAE/ANSI BACnet [27] and there are three network controller and transmitter IC vendors and hundreds of products that support it.

5.2 Network architecture

With LonWorks, the network architecture is versatile and not tightly coupled with a specific media configuration. The LonWorks network architecture consists of channels, which are physical transport media for packets. Each channel can contain a total of 32,385 nodes. In such a configuration, various media types can be used in a single network and data exchange between nodes in different channels is possible with the use of routers. In effect, the network configuration can be optimized for different network traffic at each channel. For example, in a channel where multiple low-level devices report diagnostic information at high frequency in a noisy environment, a fiber optic medium would be more suitable. For nodes outside this channel that communicate over a twisted pair medium this diagnostic information can still be acquired.

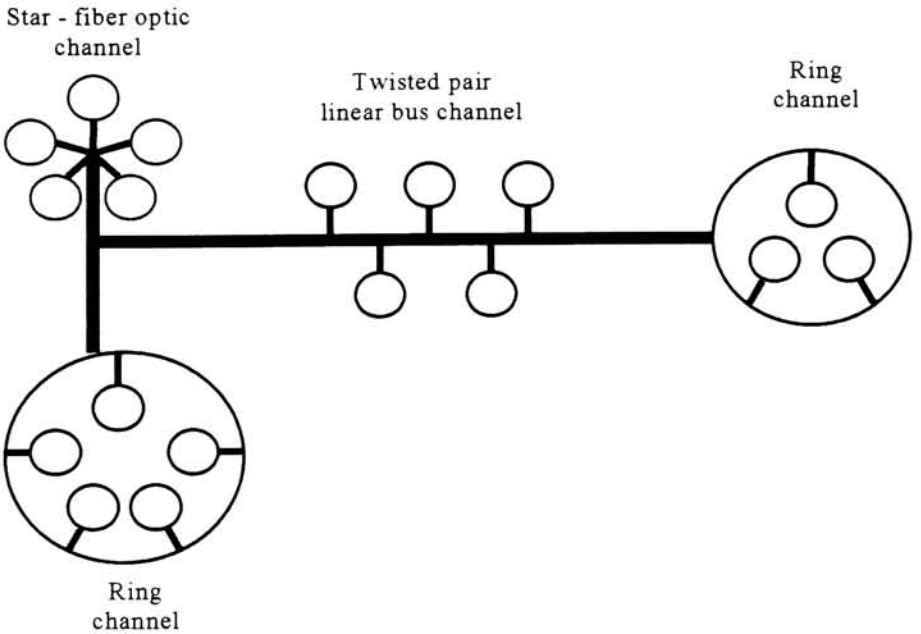

Figure 8: LonWorks Media Topology

The Physical/Media specific characteristics of LonWorks can be summarized as follows:

Signaling	EIA RS-485 compliant
Modulation	Baseband, RF (300-900MHz, IR)
Encoding	NRZ with bit stuffing
Media Coupling	DC coupled differential Tx/Rx, transformer coupled, RF
Physical media	Twisted pair, IR, fiber, power line
Isolation	500 V (optional opto-isolators on node side of transceiver)
Topology	Linear bus, ring, star.
Max. number of nodes	32,385 per domain
Data rates	78KBaud-1.25Mbaud
Max. frame length	228 bytes
Communication scheme	Master-slave, peer-to-peer.
Max. bus distance (linear bus)	2.2km for 78Kbaud, 130m for 1.25Mbaud
Max. bus line voltage	-7 to +12 V referenced to transceiver ground
Differential Output level	2.0 Volts p-p (nominal), 1.5 Volts p-p (minimum)
Network power supported	No
Hot insertion/removal of nodes	Yes
Arbitration method	Predictive p-persistent Carrier Sense Multiple Access /Collision Detection (CSMA/CD) between peers, token passing between controllers.
Error checking	16-bit CRC

Table 3: Summary of LonWorks characteristics

Notice that physical layer signaling can be based on the EIA RS-485 standard.

5.3 Communications protocol

The LonWorks communication stack is based on the ISO-OSI 7-layer model that describes the cumulative protocol structure for information transfer between two applications over a communications medium. Unlike the previous two networking protocols, LonWorks defines all of the OSI layers. In Figure 9 some layers of the OSI model are shaded similarly to signify that these groups of layers are usually handled by one processor in order to maximize device performance.

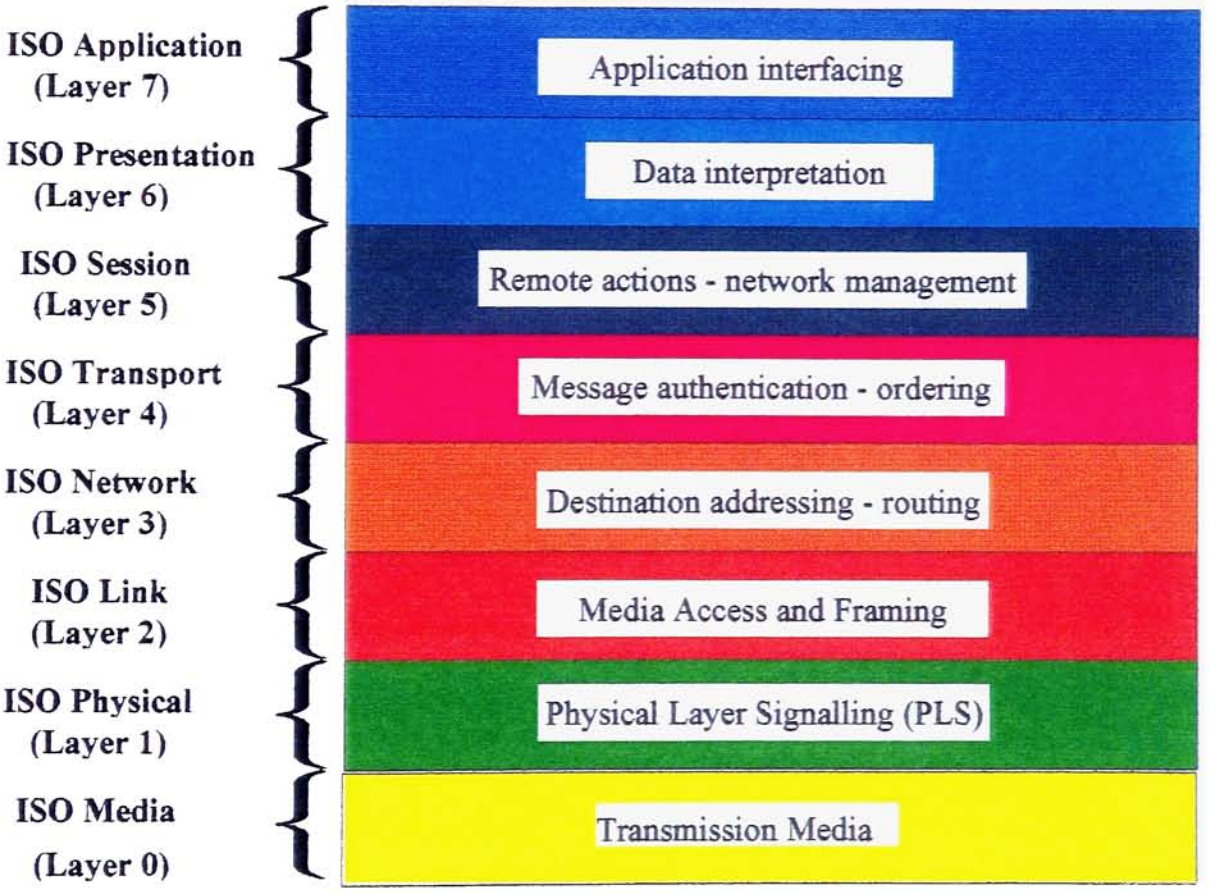

Figure 9: Mapping of LonWorks to ISO-OSI model

One of the key characteristics of this communications protocol is that it is entirely defined as an object-oriented model. Any variables or types in a node are defined as objects that are input to or outputs from the device. During network configuration, each of these static objects called *network variables* is bound to another network variable at a different node. For example, an output temperature variable in a temperature sensor device can be bound to an input temperature variable in a controller node.

Using the network variable concept, LonWorks behaves as a true distributed system with a global view of process information on the network. Network variables may be specified as authenticated (authenticated messages are used to transmit authenticated network variables). Receivers determine if a transmitter is allowed to update an authenticated network variable. In addition, a predefined priority can be associated with network variables. Finally, network variables can be either synchronous (all updates are transmitted) or non-synchronous (only the most recent value is transmitted and intermediate values between polls are ignored).

The classes of services associated with network variables are:

- **ACKD**: Acknowledge service with retries.
- **UNACKD**: Unacknowledged service.
- **UNACKD_RPT**: Unacknowledged repeated service (message sent multiple times).
- **REQUEST**: Request/Response service used for polling of network variables.

In LonWorks, a node has the ability to react on certain events that are associated with network variable updates:

- **nv_update_occurs**: A new value for a network variable has been received.

- **nv_update_fails**: Propagation of the value of an output network variable has failed.
- **nv_update_succeeds**: Propagation of the value of an output network variable has succeeded.
- **nv_update_completes**: Propagation of the value of an output network variable has completed (either successfully or unsuccessfully).

In the case of large data objects, network variables larger than 31 bytes, explicit messages can be transmitted over the network. These messages can contain up to 228 bytes and must have implicit addressing embedded in them to reach the destination node. Data exchange is carried out via explicit request - response messages. This implicit addressing is referred to as message tagging. The classes of services associated with these explicit messages are the same ones used for network variables.

5.4 Documentation and Support

Documentation for the LonWorks technology is available through distributors and suppliers of devices. Libraries of utilities and functions related to the LonTalk protocol are also readily available to network device developers. Echelon and other licensed suppliers of LonWorks equipment provide sample testing and development platforms and evaluation kits for quick product deployment.

6 Which Fieldbus is the best?

The answer to this question is simple: none. Every process application is different and has different requirements from the communications interface. As with evaluating and comparing various computer architectures, it is not simply a matter of data transfer rates or sizes.

Performance is the most important measure associated with a fieldbus but it is not a characteristic of the bus itself. Different configurations allow for different performances. Performance is typically measured by the data transfer rate of the network. Here are some equally important characteristics of a fieldbus that determine its performance:

- Redundancy of communication medium
- Network topology
- Protocol overhead
- Maximum number of network devices
- Arbitration method
- Maximum distance
- I/O update frequency (cycle time)

Networks such as PROFIBUS-DP that allow for redundant media to be distributed along with the regular communication medium are versatile and average performance is increased even at lower baud rates. In another example, performance of subnets (channels) in LonWorks that require high I/O update frequency is increased by the use of local fiber-optic star networks. It is not necessarily the properties of the fiber optic network that

increase performance but more the topology of the whole network that enables other low frequency I/O updates to take place in a different channel.

The data that is transmitted across the network contains both actual information and overhead. Overhead is any information that is only used by the communication protocol and not the application. Framing bits, device address, and fragment counters are examples of overhead information. Therefore, the ratio of actual information to transmission overhead also affects the performance of a fieldbus. Some fieldbusses compensate for large overhead by using high data rates. As an example, PROFIBUS-DP has comparable or better I/O update rates than DeviceNet, because it can operate at very high data rates, even though its protocol overhead ratio can be large.

The maximum number of devices per bus not only determines the expandability of the network but also the cycle time, which is a measure of performance. For example, the cycle time for PROFIBUS-DP, which supports up to 127 devices, would be much smaller than the cycle time for LonWorks, which supports up to 32,000 devices. This example assumes that all devices are scanned for I/O cyclically, and all devices are allocated equal periods of time for scanning.

The arbitration method is also a measure of performance because it determines the actual rate at which useful information is transferred across the network. Using this measure, DeviceNet, which uses no-penalty arbitration, can obtain a higher transfer rate than LonWorks, which uses collision resolution, and PROFIBUS-DP, which uses token passing.

Finally, performance and data integrity can decrease for longer network distances. However, flexibility in the types of communication media that can be used in a single network can compensate for this effect. PROFIBUS-DP and LonWorks are two examples

of networks that allow for fiber-optic media to be used for long distances and twisted-pair media for short distances.

To summarize, PROFIBUS-DP is a fieldbus that allows for high data rates and large packet sizes without packet fragmentation. Therefore, it proves more useful for process applications that require large amounts of sensor data to be transferred across the network at periodic intervals. DeviceNet on the other hand, allows for repeatable and deterministic performance for smaller data sizes. In addition, because of the object-oriented nature of the communication protocol, extended device diagnostics are easy to obtain. Finally, LonWorks allows for large number of nodes and subnets of dissimilar media to be connected, and provides a means for mapping process parameters to network variables seamlessly.

7 New directions

The device busses that were previously presented in this thesis constitute a new breed of industrial communications standards that implement modularity by use of object oriented technology. As a logical evolution of distributed control systems, device data is not simply a collection of I/O data buffers. Every piece of information has attributes associated with it, which provides for self-documenting distributed processes and a natural way of thinking in order to design those processes. For example, a power supply voltage readback is not simply a number but an I/O channel with units and maximum operating ranges associated with it. As seen with DeviceNet and LonWorks, this structure is available and enforced for every process component.

In DeviceNet, such components would be objects based on Analog or Digital Input or Output objects. In LonWorks, these components are *network variables* that are available anywhere on the network. PROFIBUS-FMS provides a specification for object oriented representation of process components but does not provide a real-time implementation for it. However, an object-oriented implementation of process uploading can be done in non-real time and PROFIBUS-DP can then meet the hard real time deadlines of the process.

Object-oriented methodology certainly seems to be a rightfully successful trend even in distributed control and networking applications. Although various networking standards may be based on totally different technologies, the application processes can still be the same no matter what the communications interface. The same applies to process monitoring.

Today certain architectures support the trend for object-oriented and active object structures in distributed systems; one of these architectures is **O**LE for **P**rocess **C**ontrol

(OPC). An OPC server in a microcomputer uses Microsoft's Component Object Model (COM) or Distributed COM (DCOM) to supply data to a program on the microcomputer or to a network device. Clients of the OPC model on the high end application side can include programs that OLE automation enabled such as C/C++, Visual Basic, or Excel. What this means is that a process application can be designed on a PC and communicate with another process through a well defined structure of interconnected objects that some times can be traced all the way down to the network interface (assuming object-oriented fieldbusses are used).

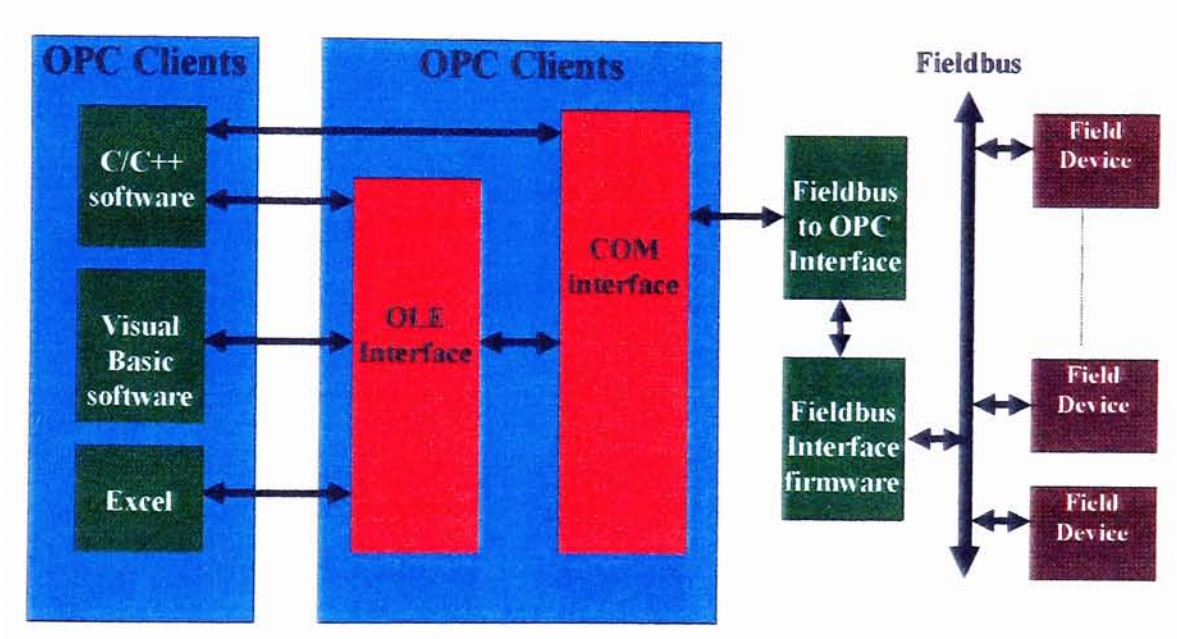

Figure 10: Application of OPC in fieldbus based process control and monitoring

Finally, another trend in industrial process monitoring is the use of JAVA, a programming language developed by SUN Microsystems. At a first glance, JAVA, with its non-deterministic non-real time behavior seems unsuitable for such a task. However, in a high level monitoring application, where only soft real time response is required, JAVA is a perfect fit because of its inherent multitasking capabilities and interface to web browsers.

A JAVA-based process-monitoring system that can be launched from web browsers can be running on a company wide version of Internet, an Intranet.

Such a system was developed for demonstration purposes in this thesis. A distributed process control system could consist of a number fieldbusses, which interconnect logical groups of devices in the system. In the proposed architecture, which is described in detail in the following sections, fieldbus controllers, often called "scanners", control each group of low-level sensor devices. These scanners also provide Internet connectivity to enable remote clients, such as JAVA applets, to modify and monitor sensor device data for any sensor device connected to a fieldbus.

8 Remote fieldbus client requirements

The end users, the system clients place certain requirements for the implementation of a remote process monitoring and control system. These requirements address the accessibility, programmability, and interfacing issues that were described in the previous sections. However, there are also considerations from the perspective of the system owner and those address issues such as security, privacy, access control, and safety. In addition, both the owner and client require the process control system to be predictable, cost effective and expandable. Predictability is necessary in order to trust experimental results. With an Internet-based remote monitoring and control system, the cost to the client would only be limited to the cost of obtaining a local Internet connection. Expandability enables less expensive and more gradual system upgrades. The remote process control system client requirements are described in this chapter.

8.1 Batch processing - Recipes

The clients of a process system (the researchers for example) can greatly benefit by being able to have unrestricted access to a certain set of fixed real-time tasks [6, p. 1401] and by being able to automate the repeated execution of those tasks by means of executing batch files. These batch files for process control are often called *recipes*. Characteristics of this form of process control are:

- Access is required not only to the complete distributed process control system, but also to individual subsets of that system.
- Access times depend on the availability of the process control system and the safety precautions that are enforced by the system's owner.

- The process system should allow scheduling of recipes not only for immediate execution but also for deferred execution.

In addition, some necessary characteristics of the individual tasks [6, p.1401] in each recipe are:

- The set of tasks that are performed during that access is usually predetermined. These tasks should execute in hard real-time even when their results will be monitored in soft real-time, locally or remotely.
- The tasks are repeated in order to average results over a long period of time.
- Tasks should map to real algorithms that researchers need to investigate.
- Granularity of the tasks must be such that:
  - low-level troubleshooting is possible but complexity is hidden from the client.
  - real-time monitoring of the results of these tasks is possible with reasonable performance. Very high granularity imposes unrealistic requirements on communications (continuous transfer of large amounts of data) [8, pp.105-106]. Some information has to be grouped so that a complete picture of the results can be obtained in real-time.

For correct and efficient use of a process control system through batch processing, clients must have access to short demonstration recipes or presentations in order to become familiar with the system remotely [6, p.1406]. In semiconductor manufacturing cluster tools for example, multiple teams of researchers could execute demonstration recipes on different clusters of the same tool concurrently. This would be possible because, in cluster tools every cluster can operate independently of all others.

8.2 Access control

Because of the increased accessibility of the distributed system, special care should be taken to make sure that certain access restrictions are enforced [6, p.1401]. Reliable and safe real-time operation requires that no two clients can directly control a system simultaneously. This, known as *access control*, should:

- be effective during both real-time operation and scheduling of batch files for deferred execution.
- allow the client to be informed if the requested deferred execution (batch file of tasks) cannot be scheduled during the requested time or day due to pre-existing reservation.
- maintain a persistent database of batch file requests from the clients.
- implement security measures to enforce different access restrictions for different users of the system. External access restrictions must also be enforced for safety reasons.

8.3 Prioritized scheduling

In any manufacturing or research facility, there are several different levels of priority assigned to various personnel. For example, during high volume production or marathon testing, research personnel has a lower access priority than production personnel. The process control system should, in that case, be able to dynamically assign and re-arrange access priorities for real-time and batch tasks according to directives from the system owner. In general priority scheduling is necessary for:

- more critical projects – research
- production deadlines
- system upgrades or maintenance
- optimization of batch (recipe) task execution

8.4 Modularity, flexibility, autonomy

Research and Development groups should have the ability to reprogram or reconfigure part of a process control system on the fly without having to disrupt the operation of other sections. This is only possible if the system is modular and flexible and parts of the system can operate autonomously.

- **Modularity** refers to how the distinct functions of the system are structured as individual devices in the system or part of the system.
- **Flexibility** refers to the degree of change the system can sustain and still produce the expected result.
- **Autonomy** of a system refers to the degree that clusters of a system can function independently, for normal operation, testing or maintenance purposes.

Ideally, for advanced programmability, developers or researchers could have access to a standardized Applications Programming Interface (API) of system functions that can remotely programmed in the form of a software module (ex. functions written in C). This API would, in essence, provide transparency between the networking details encapsulated within the system itself and the client task requests [6, pp.1401, 1402, 1406].

8.5 Standardized interface

Graphical User Interfaces (GUI) for individual devices in a process control system enable a transparent interface to any device in the system. GUIs for clusters in the system serve as Man-Machine-Interfaces (MMI) for external operation of the entire system by trained personnel. With current advances in distributed objects (OLE and ActiveX) and the popularity of Internet-based applications, GUIs can be developed to have the following properties:

- Local or remote operation via a common standardized interface. JAVA applets or standalone ActiveX components can be used as remote device or system interfaces.
- Selective information processing to maximize performance. This relates to the granularity of the information that is retrieved from the process system as described in section 8.1.
- Unambiguous display of critical alarm, system health, and system efficiency information with priority over other monitoring functions.

8.6 Data logging

Remote data logging proves to be very useful in situations where multiple systems are located at distant and separate locations and are in grave need of troubleshooting. Semiconductor tool manufacturers are very aware of this since most of this equipment, although manufactured in the United States, is operating in South East Asia. However, even for moderately complex systems, there could be a very large number of data that a remote monitoring/data logging system would have to receive. Desirable properties of a data logging utility would be:

- Data logging must be performed at the location where the actual process control system is operating so that information can be recorded in hard real-time. If information were to be recorded by a remote client, communication delays between that client and the server would affect the frequency of data logging steps.
- The process control system must have access to a large enough amount of storage space (random access memory, storage media) to enable data logging over an extended period of time. Troubleshooting of intermittent operation often requires data logging for hours or even days.
- A specialized data transfer protocol should be used to enable the client to download large data logs efficiently. All other remote client operation and monitoring functions should be disabled (client interface should be off-line) during downloading of data logs.

9 Proposed system

9.1 System architecture

One approach in achieving the above functionality is the use of a World Wide Web based client server model for the monitoring and control system. The system architecture is shown in Figure 11.

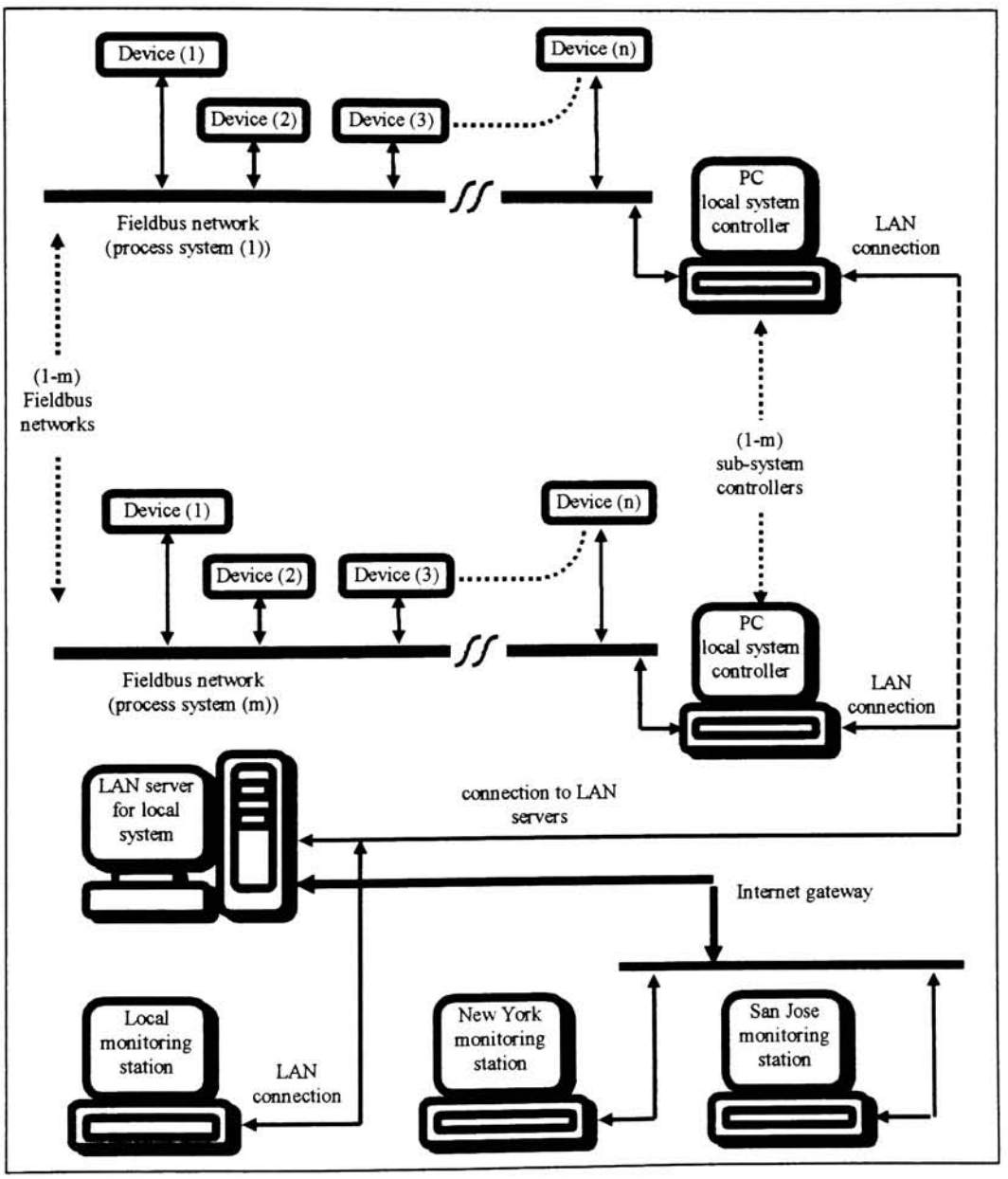

Figure 11: Remote Process Monitoring and Control System architecture

- In the above figure, each of the local system controllers is connected to a subset of the entire process control system by means of a high-speed digital network, a fieldbus.
- On the same station, a server application is running to make the local fieldbus data available to any interested clients. In essence, the server controls the access points of any external clients and enforces security at the same time.
- Client applications can run on any computer connected to the plant Local Area Network (LAN), as well as the computer where the server is residing. Apart from Intranet clients, a local gateway from the LAN server to the Internet makes worldwide client access possible through the same interface.
- Monitoring stations can run both client and server applications to enable peer to peer communications between stations.
- Extra security is enforced in a station by station basis by shutting down any monitoring application servers that are off-line or restricted to plant accessing only.

To achieve a uniform interface in Intranet and Internet access, JAVA is the language chosen for client applications in the proposed system. JAVA is non-deterministic by nature but monitoring applications do not have hard real-time requirements and control parameters can be downloaded to a system during initial configuration. For the interface of the server application to the fieldbus network interface (network data acquisition card) an initial implementation in Visual Basic is provided. This Visual Basic application acts as an application gateway between an Intranet or the Internet and a fieldbus-based control system.

9.2 Overview of the Client/Server interface

The remote control and monitoring system uses the client/server architecture over a connection-oriented messaging stream to access a device from a remote location. The server is the device that physically controls the remote network and at the same time accepts requests from remote clients. A server will not send any messages unless it is requested to do so. The clients on the other hand, can initiate a request that will be sent to the server and usually, but not always, will wait for an appropriate response. In essence, the clients will act as the connection masters while the server will be the connection slave.

Messages will be divided as *requests* and *responses*. Both message types are defined from the perspective of the remote client who intends to acquire data and initiate actions on the local system. Therefore, *requests* are messages sent from the client to the server and *responses* are messages sent from the server to the client. Figure 12 shows the flow of data across an established connection.

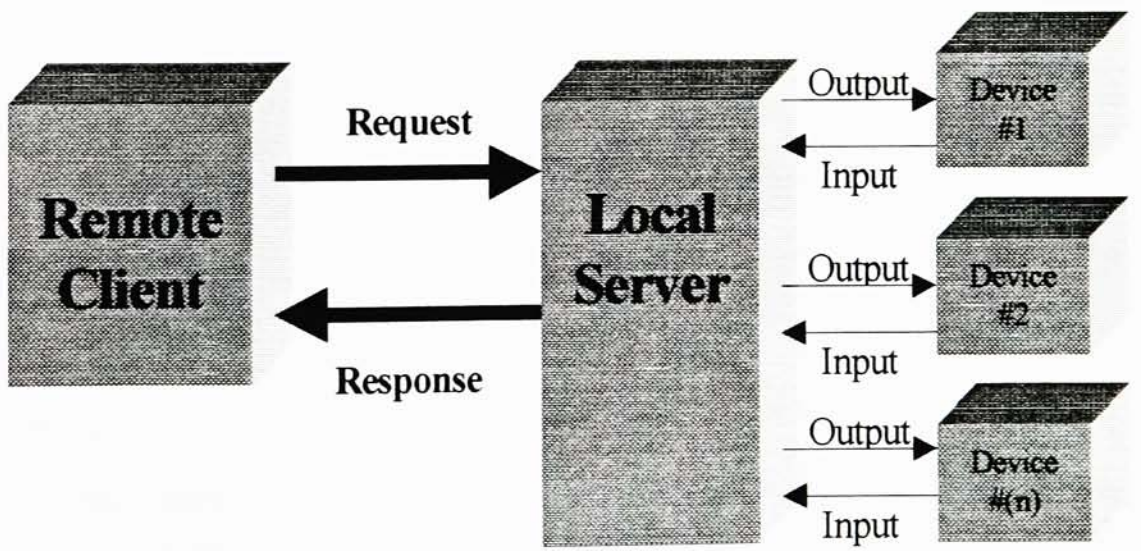

Figure 12: Client-Server message transfer

For every device that the server monitors/controls there is a set of input/output data that the device can continuously provide/process. Again, for consistency, the terms *input* and *output data* are used as seen from the perspective of the server.

9.3 Communications protocol between client and server

A proprietary communication protocol was developed to enable communication between the server and a client in this system. Message packets between the server and any client are essentially strings of characters travelling from a server socket connection to a client socket connection. Every field (which is a set of characters by itself) is delimited from the next field by a space character and a carriage return character delimits the end of the packet. All fields that represent numerical values are formatted as decimal numbers. The most significant digit of these numbers always precedes the least significant digits.

Every message between the server and a client – in either direction – has the following format:

MSGID	SP	**ARG0**	SP	**ARG1**	SP	.....	SP	**ARGn-1**	SP	**CHKSUM**	CR

- MSGID

  It is three characters long and represents the number of the remote command that will be executed. The number of possible commands is equal to the number of possible 3 – alphanumeric digit combinations (36 choose 3).

- ARGx (x = 0, 1, ..., n-1)

  These are the arguments that the remote command will use. Every command interprets its arguments differently. For the currently supported commands, all arguments are interpreted as decimal - numeric values that are received digit by digit. For example, if the value of arg0 is 120, three characters will be received for arg0: '1', '2' and '0' When a command will use a combination of arguments as one argument, the first argument is always assumed to be the highest order byte. For example, a two-byte integer is transmitted as two arguments, where the first argument in the list is the most significant byte and the next argument is the least significant byte. When the ASCII digits for these numbers are received by the destination, they are converted to hex values and treated as bytes.

- CHECKSUM

  This is an additive, modulo 256 checksum of the other values in the packet. Its format is the same as the format of the arguments; i.e. it is interpreted as a decimal – numeric value that is read digit by digit from a character stream.

The following message packet formats are presently defined as valid requests:

ID:
"PGS"

Message Attributes	**Attribute description**
Remote Call Name	*PingServer()*
Type (Req/Resp)	Request/Response
Min. length (characters)	1 for Request – 1 for Response
Arguments (ARG0 to ARGn-1)	None.
Action (at server)	Server sends a PingServer() response message as an acknowledgement.

ID:
"XDD"

Message Attributes	Attribute description
Remote Call Name	*ExchangeDeviceData()*
Type (Req/Resp)	Request/Response
Min. length (characters)	6 for Request – 5 for Response
Arguments (ARG0 to ARGn-1)	Client sends command bytes to server and requests readback bytes from server • ARG0 = Address of device on the network that is controlled by the server. • ARG1 = A variable message length depending on the size of the outputs • ARG2 – ARGn-1 = An argument list consisting of the output data for the remote device, starting with the least significant output byte.
Action (at server)	Server sends a message containing: • A message ID identical to the request message ID (ID = "XDD") • ARG0 = Address of device on the network that is controlled by the server. • ARG1 = A variable message length depending on the size of the inputs • ARG2 – ARGn-1 = An argument list consisting of the input data from the remote device, starting with the least significant input byte. • A checksum of the response message. On error, Server sends an error response message with a predefined error code in ARG0 (see *Error Codes*)

ID:
"RDR"

Message Attributes	Attribute description
Remote Call Name	*RunDeviceRecipe()*
Type (Req/Resp)	Request/Response
Min. length (characters)	6 for Request – 1 for Response
Arguments (ARG0 to ARGn-1)	Client sends recipe bytes to server and requests to start executing the recipe • ARG0 = Address of device on the network that is controlled by the server. • ARG1 = The number of recipe steps that are uploaded to the server • ARG2 = A flag that indicates if the recipe is repeating, i.e. when the last step is reached, the recipe continues executing at the first step. A value of '1' indicates that the recipe will be repeating. A value of '0' indicates that the recipe will stop executing when the last step is reached. • ARG3 – ARGn-1 = An argument list consisting of the recipe steps in the following format: ARG3 = step(0).value.LSB ARG4 = step(0).value.MSB ARG5 = step(0).duration ARG6 = step(1).value.LSB ARG7 = step(1).value.MSB ARG8 = step(1).duration --- The step values are the setpoints for the variable that the recipe updates. The step durations represent the duration of each step in milliseconds. • A checksum of the response message.
Action (at server)	Server sends a message with • ID identical to the request message ID (ID = "RDR") as an acknowledgement On error, server sends an error response message with a predefined error code in ARG0 (see *Error Codes*)

ID:
"SDR"

Message Attributes	Attribute description
Remote Call Name	*StopDeviceRecipe()*
Type (Req/Resp)	Request/Response
Min. length (characters)	1 for Request – 1 for Response
Arguments (ARG0 to ARGn-1)	Client sends requests to stop executing the recipe • ARG0 = Address of device on the network that is controlled by the server. • A checksum of the response message.
Action (at server)	Server sends a message with • ID identical to the request message ID (ID = "SDR") as an acknowledgement On error, server sends an error response message with a predefined error code in ARG0 (see *Error Codes*)

ID:
"TDR"

Message Attributes	Attribute description
Remote Call Name	*ResetDeviceRecipe()*
Type (Req/Resp)	Request/Response
Min. length (characters)	1 for Request – 1 for Response
Arguments (ARG0 to ARGn-1)	Client requests to reset the currently executing recipe to step 0. • ARG0 = Address of device on the network that is controlled by the server. • A checksum of the response message.
Action (at server)	Server sends a message with • ID identical to the request message ID (ID = "TDR") as an acknowledgement On error, server sends an error response message with a predefined error code in ARG0 (see *Error Codes*)

ID:
"RDD"

Message Attributes	Attribute description
Remote Call Name	*RunDeviceDatalog()*
Type (Req/Resp)	Request/Response
Min. length (characters)	6 for Request – 1 for Response
Arguments (ARG0 to ARGn-1)	Client requests to start data logging for a certain number of steps and step duration • ARG0 = Address of device on the network that is controlled by the server. • ARG1 = The number of data logging steps • ARG2 = The duration of each step in milliseconds. • A checksum of the response message.
Action (at server)	Server sends a message with • ID identical to the request message ID (ID = "RDD") as an acknowledgement On error, server sends an error response message with a predefined error code in ARG0 (see *Error Codes*)

ID:
"SDD"

Message Attributes	Attribute description
Remote Call Name	*StopDeviceDatalog()*
Type (Req/Resp)	Request/Response
Min. length (characters)	1 for Request – 1 for Response
Arguments (ARG0 to ARGn-1)	Client requests to stop data logging • ARG0 = Address of device on the network that is controlled by the server. • A checksum of the response message.
Action (at server)	Server sends a message with • ID identical to the request message ID (ID = "SDD") as an acknowledgement On error, server sends an error response message with a predefined error code in ARG0 (see *Error Codes*)

ID:
"GDD"

Message Attributes	Attribute description
Remote Call Name	*GetDeviceDatalog()*
Type (Req/Resp)	Request/Response
Min. length (characters)	6 for Request – 8 for Response
Arguments (ARG0 to ARGn-1)	Client requests to receive the latest data log information • ARG0 = Address of device on the network that is controlled by the server. • A checksum of the response message.
Action (at server)	Server sends a message containing: • A message ID identical to the request message ID (ID = "GDD") • ARG0 = Address of device on the network that is controlled by the server. • ARG1 = A variable message length depending on the size of the inputs (readbacks) • ARG2 = The number of completed data logging steps (not necessarily equal to the number of steps specified by the *RunDeviceDatalog()* command) • ARG3 = The duration of each step in milliseconds. • ARG4 – ARGn-1 = An argument list consisting of the input data from the remote device for each data log step, starting with the least significant input byte. ARG4 = step(0).deviceByte(0) ARG5 = step(0).deviceByte(1) --- ARG(4+n) = step(0).deviceByte(n) ARG(4+n+1) = step(1).deviceByte(0) ARG(4+n+2) = step(1).deviceByte(1) --- • A checksum of the response message. On error, Server sends an error response message with a predefined error code in ARG0 (see *Error Codes*)

A set of error codes is defined for the purpose of informing the client of a failure in achieving the desired result from a remote command call. The following error codes appear in ARG0 of the response message that is received from the server:

Error Code	Error Description
0	No Error
1	Remote network is not on-line.
2	Remote device is not being scanned.
3	Remote device cannot be reached.
4	Server received a wrong value for the length of device outputs.
5	Recipe already running. Stop current recipe first.
6	Cannot stop recipe. No recipe running.
7	Cannot reset recipe. No recipe running.
8	Data logging already running. Stop current data logging first.
9	Cannot stop data logging. Data logging is not running or is finished.
10	Cannot send data log. No data logging was initiated.

10 Conclusions

Fieldbus networks have recently emerged as a powerful alternative to conventional methods for real-time process control. They allow for the design of modular and reliable distributed control systems, while providing cost effective solutions for upgrading from older technology. This thesis described an approach to remote operation and monitoring of process control systems, which use fieldbusses to interconnect all low-level sensor devices. The main focus was placed on achieving real-time performance in process monitoring and operation over a framework of non-homogeneous networks with different degrees of reliability in communications. In particular, the remote Internet access to fieldbus-based control systems was studied. For demonstration purposes, a Visual Basic implementation of an Internet server for a DeviceNet fieldbus, and a JAVA implementation of a remote device client were developed and tested.

This thesis has described several advantages of fieldbus networking and has introduced concepts in real-time communications and task scheduling. In addition, general requirements for communications in real-time distributed systems were presented. These requirements do not only apply to networking within isolated process control systems, but also to the inter-networking of groups of such systems. Furthermore, this thesis presented a set of desired system client properties for reliable telemonitoring and teleoperation. The design of a remotely accessible process control system should allow for multiple clients to interact with such a system in a safe and user-friendly manner.

To illustrate the use of fieldbusses in real-time process control systems, the technical characteristics for three popular fieldbus networking schemes were presented. These fieldbusses were then compared to demonstrate the strengths and weaknesses of each fieldbus. More importantly, they were compared to demonstrate that the use of a particular fieldbus in the proposed architecture would not require any changes in the client-server interface.

The final sections of this thesis presented the client-server architecture for a remote process operation and monitoring tool. The protocol for client-server communications was described and several client requirements that enable real-time behavior were analyzed. Requirements such as remote recipe submission, remote data logging and real-time graphing of device readback values were implemented as client utilities. To demonstrate the principles presented in this thesis, detailed operation and configuration information was presented for a sample DeviceNet server and a DC-plasma power generator.

10.1 Problems encountered

During the course of this thesis a few minor problems were encountered. First, only very recent publications included information for fieldbusses in process control systems. In addition, articles on telemonitoring and teleoperation were mostly theoretical and did not provide any examples of real-time performance. Telemonitoring and teleoperation has traditionally been used in robotic systems but not as much in process control systems, although the similarities between such real-time systems are many.

Another minor problem that was encountered during the implementation of the demonstration system was the errors associated with the graphics toolkit in older versions of JAVA (JDK 1.1.2). Some of the software had to be modified to enable migration to the newer version of JAVA (JDK 1.1.5). A drawback in this migration was that the client interface could not be invoked from old web browsers because they do not support this new version of JAVA. In addition, the client interface could not be invoked when the server was connected to an Intranet where a firewall was active. Common configurations for firewalls usually allow for http or e-mail connections only. The client-server connection in this demonstration system uses socket connections through ports that such a firewall would not allow.

Finally, the procedure for exchanging device I/O between the client and the server was recently modified. Previously, the client was running one thread, which only requested data from the server for a fieldbus device. The client transmitted the command data for that device only when an event occurred at the GUI (button pressed, text box data changed, etc.). This created conflicts between the fast thread that was requesting data and the less frequent GUI events that were causing command data to be transmitted. The problem was fixed by sending command data to the server as part of the message for requesting readback data. This method allowed for reliable and synchronized data exchange between the client and the server.

10.2 System performance

In the demonstration system, the server was connected to the Internet through a private Intranet and the system was tested with two remote fieldbus devices. The client interfaces continuously scanned for I/O every 100 milliseconds. Various update intervals ranging from 1 millisecond to 1 second were tested. The performance of the client-server connection appeared to deteriorate significantly for update intervals below 50 milliseconds. This behavior was due to the fact that messages transmitted over the Internet have variable delivery delays. For repeated messages of the same type (i.e. device I/O data exchange) duplicate reception did not appear to be an issue since all duplicates were simply ignored on both the client and the server. However, when the client used a very high I/O scan rate and, at the same time, transmitted data for GUI events (i.e. remote recipe submission), the system experienced problems due to communication synchronization.

Furthermore, since the lower bound of I/O updates through DeviceNet (in this server implementation) is 5 milliseconds, the server could not respond to client requests at intervals lower than 5 milliseconds. The performance of client-server communications was

measured by using software timers on both the client and the server receiving queues. The test results have shown a +/-10% variation in message delivery for update intervals of 100 milliseconds or higher.

Certainly, although this performance is more than adequate for the test system, it could degrade for a number of devices, which is closer to the limit specified by the fieldbus in use. Even in that case, the impact on the real-time performance of the process control system would be significantly less. In this system, the client did not directly control all major functions that required hard real-time performance. Utilities such as remote process recipe and data logging proved to be a very useful tool in achieving such a real-time performance. They allowed for transfer of hard real-time functionality to the fieldbus controller, which enforces deterministic access to sensor devices. Furthermore, remote recipes allowed for automation of device operation and testing while they enabled accurate experimentation with repeatable results.

In addition, to provide more accurate monitoring information and avoid issues related to communication latencies, a data logging utility was used. This utility, combined with remote process recipes, allows the client to submit tasks for a specific device to the server and achieve deterministic performance in controlling that device, as well as obtain accurate monitoring information. Recorded status information from tests that used a combination of these two utilities verified the above observation.

10.3 Suggestions for improvement

Several enhancements could be applied to both the client and server software to improve its reliability as well as to add more functionality. The client and server software could be modified to provide an additional degree of fault detection in communications. In that case, certain conditions such as message delivery timeouts, message retransmission,

duplicate message detection, and client-server disconnection notices should be detected and handled.

In addition, to avoid impact on real-time performance during transmission of large packets (i.e. process recipes, data logs), a file transfer protocol could be used to transfer this data over a separate socket connection. This would simplify message transmission scheduling by using separate threads for each connection in both the client and the server software.

Robustness can be enhanced in the current implementation by adding software that handles access restrictions for multiple device clients. In essence, the fieldbus server would not only allow one client to operate a device, but also multiple clients to monitor that device. In addition, access restrictions could be a function of safety, security and operation or maintenance schedule for the process control system.

10.4 Future study

The principles presented in this thesis could be applied to the development of a complete remote operation and monitoring tool for multiple groups of process control systems. Currently, the demonstration software only allows individual device clients to remotely operate components in a process control system. Further work could result in the development of a tool that allows one client to remotely configure, operate and monitor an entire process control system. In addition, remote operation could be automated by using process recipes, which combine parameters not only from a single device, but also multiple devices in the entire system. This would actually enable the process control system to be modeled as a global set of parameters in a distributed environment.

11 Project implementation resources

- *DeviceNet*™ master network card (local system controller) with MS-Windows software driver libraries (DLL).
- *DeviceNet*™ slave network cards to implement fieldbus network and perform remote I/O acquisition (simulate real-time manufacturing process system).
- *DeviceNet*™ cabling and hub equipment for implementing fieldbus.
- Visual Basic and JAVA compilers for generating acquisition and monitoring software.
- Visual Basic ActiveX controls for socket connectivity (Catalyst SocketWrench).
- Two or more computer stations (PC, SPARC, etc.) with connection to the Internet.

12 References

[1] Amund Aarsten, David Brugali and Giuseppe Menga, "*Designing Concurrent and Distributed Control Systems*", Communications of the ACM, Vol. 39, No.10, pp. 50-59, Oct. 1996.

[2] M. Frans Kaashoek, Robert Van Renesse, Hans Van Staveren and Andrew S. Tanenbaum, "*FLIP: An Internetwork Protocol for Supporting Distributed Systems*", ACM Transactions on Computer Systems, Vol. 11, No. 1, pp. 73-106, Feb. 1993.

[3] Krithi Ramamritham, John A. Stankovic, "*Scheduling Algorithms and Operating Systems Support for Real-Time Systems*", Proceedings of the IEEE, Vol. 82, No. 1, pp. 55-67, Jan. 1994.

[4] K. M. Zuberi, K. G. Shin, "*Real-time decentralized control with CAN*", Proceedings 1996 IEEE Conference on Emerging Technologies and Factory Automation (EFTA '96), Vol. 1, pp. 93-99.

[5] K. P. Birman, "*The process group approach to reliable distributed computing*", Communications of the ACM, Vol. 36, No. 12, pp. 37-53, Dec. 1993

[6] J. L. Zhen, M. A. Lewis, K. Tan, "*Towards Universal Access to Robotic Resources*", Proceedings of international conference on Intelligent Robots and Systems (IROS 1996), Vol. 3, pp. 1400-1407.

[7] E. Sisa, "*Communication systems to monitor and control the production environment*", Proceedings 1997 IEEE Annual Textile, Fiber and Film Industry Technical Conference, pp. 1-7.

[8] Y. Wakita, S. Hirai, K. Machida, "*Intelligent monitoring system for limited communication path: telerobotic task execution over Internet* ", Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 2, pp. 104-109.

[9] "ODVA *DeviceNet*™ Specifications", Vol.1 Release 2.0, Vol.2 Release 2.0

[10] ODVA home, http://www.odva.org

[11] Deutsche Industrie Norm, "*PROFIBUS Standard DIN19245* ", *Parts 1-3*, Beuth-Verlag, Berlin, 1989-1990.

[12] Klaus Bender, "*PROFIBUS The Fieldbus for Industrial Automation*", Prentice Hall, 1993.

[13] PROFIBUS User Organization, http://www.profibus.com

[14] MOTOROLA, "*LonWorks Technology Device Data*", Rev 2, Q4/96.

[15] ECHELON, "*LonWorks Networks Reference and Developer's Preview*", 1996.

[16] A. Tanenbaum, "*Computer Networks*", pp. 28-44, pp. 252-253, pp. 577-620, pp.622-628, pp. 681-720, Prentice Hall, 3rd Edition, 1996.

[17] G. Coulouris, J. Dollimore, T. Kindberg, "*Distributed Systems: Concepts and Design*", pp. 99-119, pp. 127-138, pp. 353-370, pp. 477-513, Addison Wesley, 2nd Edition, 1994.

[18] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "*Design Patterns: Elements of Reusable Object-Oriented Software*", pp. 107-134, Addison Wesley, 1995.

[19] OPC Programmer's Connection, http://dspace.dial.pipex.com/opc

[20] Merlin and Conrad Hudges, "*JAVA Network Programming*", pp. 9-31, pp. 123-152, pp. 288-321, pp. 350-368, Manning Publications, 1997.

[21] Doug Lea, "*Concurrent Programming in JAVA*", pp. 121-212, Addison Wesley Longman Inc. 1997.

[22] E. Simon, "*Distributed Information Systems: From client/server to distributed multimedia*", pp. 35-36, pp. 50-77, pp. 359-377, McGraw-Hill, 1996.

[23] MKS Instruments, *"The Sensor Bus Effort: Importance of; Background; Trends"*, pp. 8-9, pp. 16-17, 1997.

[24] Robert Bosch GmbH, *"CAN Specification Version 2.0"*, Postfach 50, D-7000 Stuttgart, 1991.

[25] *"Road vehicles - Interchange of digital information - Controller area network (CAN) for high-speed communication"*, ISO-11898, ISO, 1993.

[26] *"Road vehicles – Low-speed serial data communication"*, Parts 1-3, ISO-11519, ISO, 1994.

[27] ASHRAE/ANSI, *"Standard 135-1995 - BACnet - A Data Communication Protocol for Building Automation and Control Networks"*, 1995.

[28] Electronic Industries Association (EIA), *"EIA RS-485 – Standard for electrical characteristics of generators and receivers for use in balanced digital multipoint systems"*, 1983.

APPENDIX A: Server application user manual

Server configuration

To install the DeviceNet bus communications, first install the SST DeviceNet board in the PC computer and the necessary SST Scanner Demo for Windows software following the installation instructions that are provided by SST. This section will describe a procedure for configuring the server application program.

First, run the DeviceNet Server program. When the *Dnsw32 – Interface Setup* message box appears, make sure the correct driver name for the interface card is displayed, put a check mark in the *overlap enabled* check box and click *OK* to continue. At this point the Server Scanner window should be as follows:

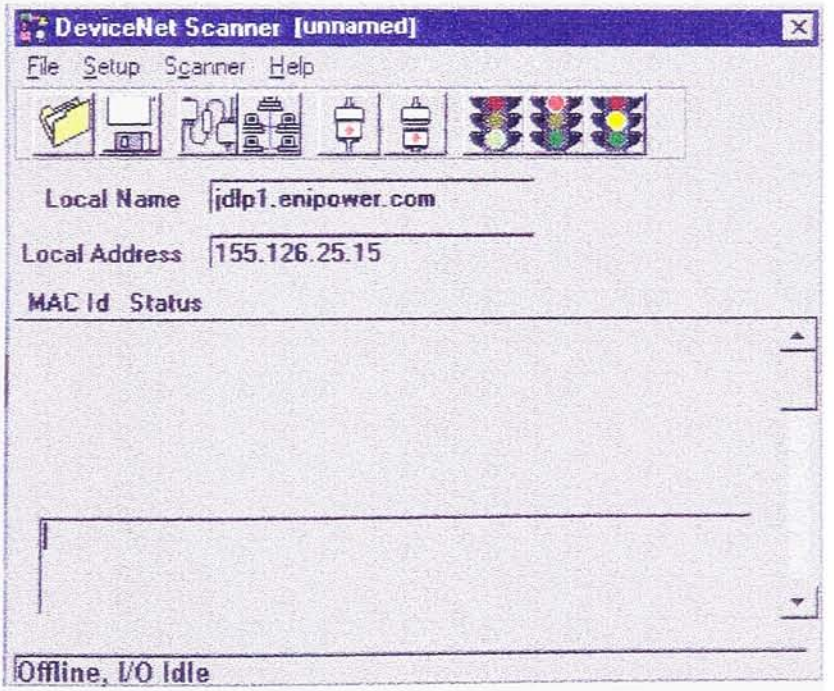

Figure 13: Server interface – DeviceNet scanner not configured

The DeviceNet scanner is off-line and no devices are scanned for data. Also, the main window does not display a scan list, which is a list of devices to be scanned for data. However the server is on-line and can respond to IP requests at the address shown in the **Local Address** box.

To create a scan list, perform the following steps:

1. Click on ***Setup*** on the main menu bar
2. Within the ***Setup*** menu select ***Scanner***. This will bring up a new window that enables the user to input information about all the devices on the network. That window should be as follows:

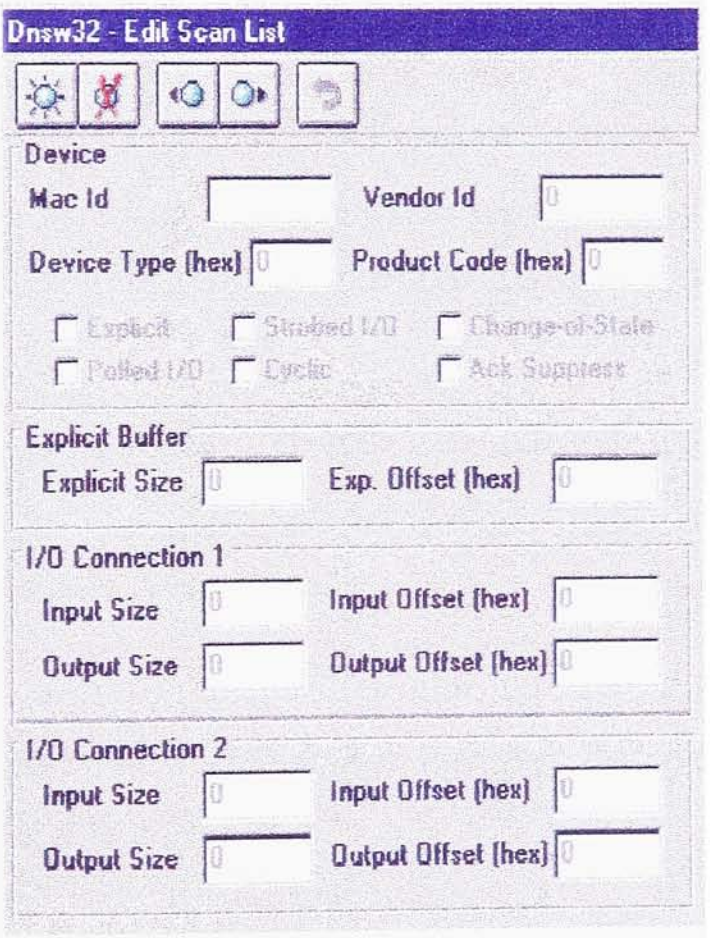

Figure 14: Server interface – empty DeviceNet scanlist

The first button at the top left corner is used to add a new device to the network scanlist. This is the new device button. The next button is the delete device button, which deletes the currently displayed device. The next two buttons are used to move to the previous device and next device in the scanlist respectively. Finally, the last button is the undo button.

3. Make sure that the actual DeviceNet device that is being added here is correctly configured for network address and baud rate (address and baud rate switches at the device are correctly set). To set the bus address for the device, choose a unique address in the range of 1-63 and set the address switches accordingly. Address 0 is usually selected for the master DeviceNet card (which is controlled by this server application) so we exclude that from the range of available slave addresses.

4. In the window of step 2, click on the new device button and in the message box, set the **Mac Id** value to be the same as the address of the DeviceNet device that was just configured in step 3. In this example, we use Mac Id (bus address) 10. Click on **OK** in the message box. Back in the window of step 2, leave the rest of the text boxes set to 0 and make sure the **Explicit** and **Polled I/O** checkboxes are enabled.

5. In the same window, and in the Explicit Buffer block, set the **Explicit Size** to 16 and the **Exp.Offset (hex)** to 1300. This application stores communication buffers at locations specified in this window. We will use locations starting at 1300 (hex) and the size for each buffer will be 100 (hex). Make sure that the buffer locations are unique for each connection and every device.

6. In the same window, and in the I/O Connection 1 block, set the **Input Size** to 9, the **Input Offset (hex)** to 1400, the **Output Size** to 5, the **Output Offset (hex)** to 1500.

By setting the offsets to 1400 and 1500 we make sure that the buffers are at unique locations. Leave the rest of the input form as is. The final result should be as follows:

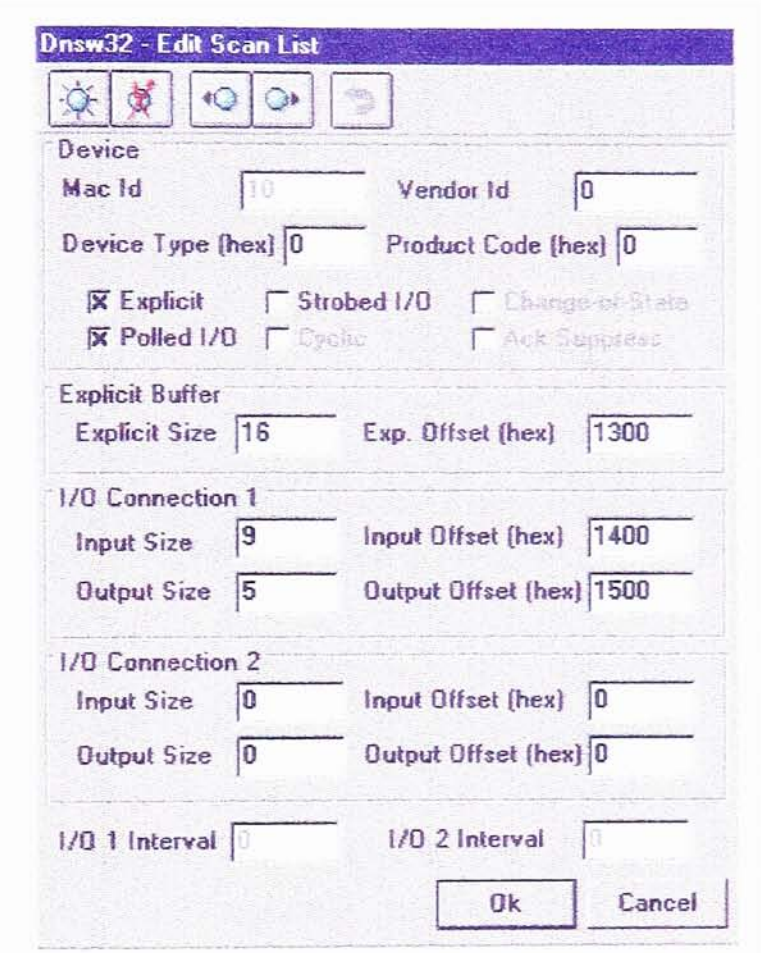

Figure 15: Server interface - adding device in DeviceNet scanlist

7. Repeat the process for more devices by clicking on the new device button. Note that once you move to another device, the information for previous devices cannot be modified. Once all the network devices have been configured, click on **OK** to return the main application window. An example of what that window would look like if several devices had been configured is:

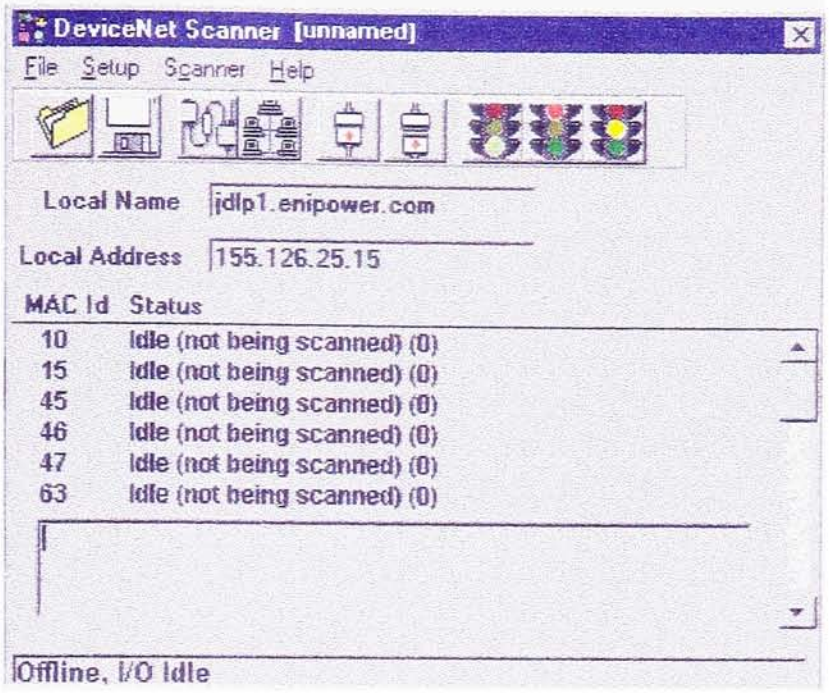

Figure 16: Server interface - DeviceNet scanner configured

All device MAC Ids (network addresses) and their status is now available. The **(0)** at the right most side of the status of each device is a fault-transition counter. When the network is scanned, the devices transition into the Active status. If a fault occurs at a device or the device is not responding, this counter will increment to 1. Every time the state of a device is transitioned from Active to Timeout the counter will increment by one.

8. To save the current scan list and be able to use it at a later time, click on ***File*** at the top menu bar and select ***Save As*** within that menu. Enter the name of the file the scan list will be save in and make sure the file extension is *SCL*. To load the same scan list in the future click on ***File*** at the top menu bar and select ***Open***.

9. Proceed by setting the DeviceNet master to the on-line mode. To do that, select ***Scanner*** from the top menu bar and within that menu select ***Go Online***. A new window will pop up:

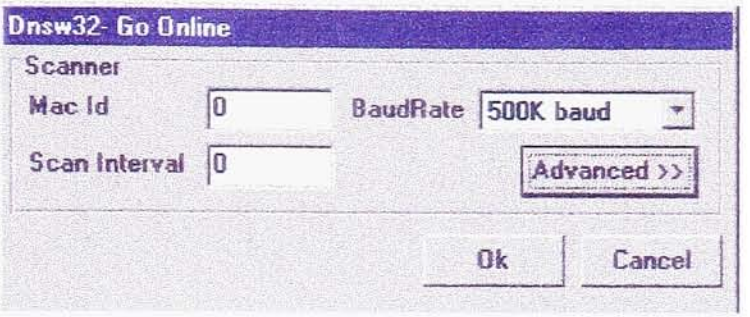

Figure 17: Server interface – *"Go-Online"* function in DeviceNet scanner

In the **Mac Id** text box, enter the address of the master DeviceNet board in the PC. This could be any address in the range 0-63 but the preferred address is 0. **Scan Interval** is the number of milliseconds between scan cycles and for fastest scans, it should be left at 0. Finally, the baud rate displayed in this window and the one selected by rotary switches for all the network devices should match. This is the baud rate that the master device will use to communicate to all the devices in the main application window. The preferred baud rate is 500 KBaud.

10. Click on **OK** in this window and confirm that DeviceNet can go-online by clicking **YES** in the next message box to acknowledge. The status bar in the bottom of the main window should display *500K, Inactive, I/O Idle* as in the following window:

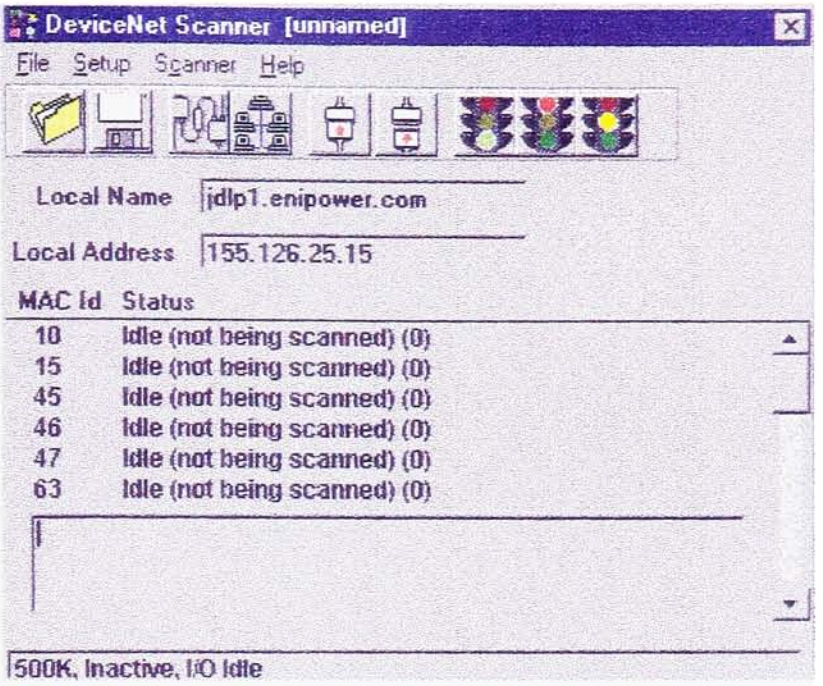

Figure 18: Server interface - DeviceNet scanner online and inactive

11. The scanner is now on-line but not performing and I/O scans. To start the scanner and exchange I/O click on ***Scanner*** at the top menu bar and then click on ***Start Scan*** within the Scanner menu. Then, within the same Scanner menu, click on ***IO Active***. Assuming only our device with MAC Id 10 is connected to the network, the main application window should be:

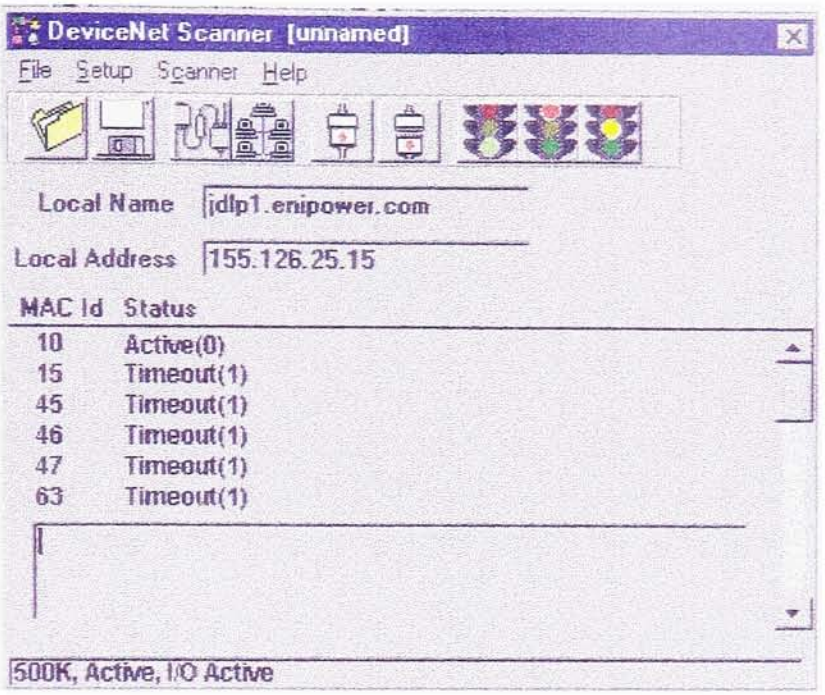

Figure 19: Server interface - DeviceNet scanner online and active

In this window, our device with MAC Id 10 is *Active(0)*, which means that it is being scanned for I/O and there were zero timeouts since the scan started. The rest of the devices are timed-out either because they are physically disconnected from the network or they are not powered up. They have all timed-out once since the scan started.

12. Next, we can modify the I/O data for a device locally. To view the data click on ***Scanner*** at the top menu bar and then click on ***Device I/O Data*** within the Scanner menu. The following window should come up:

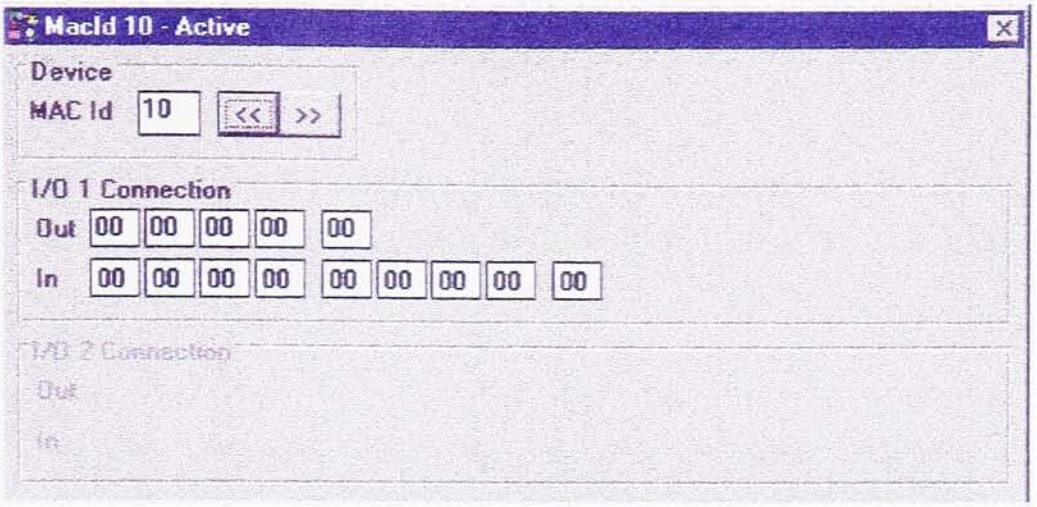

Figure 20: Server interface - DeviceNet device I/O data

This window indicates that there is only one connection available for the device with MAC Id 10 (which is what we configured it for). **Out** is the set of scanner output data. This data is sent to the actual device (commands). **In** is the set of scanner input data. This data is received from the actual device (readbacks). All data is displayed as bytes in hexadecimal format. We refer to these data bytes as **Output1 – Output5** and **Input1 – Input9** (reading from right to left). For example, the output data byte next to the **Out** label is **Output5**, the next one **Output4** and so on.

At this point, the DeviceNet scanner is configured and scanning the network for I/O data. In addition, the application is listening for IP requests from remote clients.

APPENDIX B: Client application user manual

This remote client interface demonstrates the operation and monitoring of a DC plasma power generator. The remote interface, which is displayed in the following figure, is a JAVA applet.

Figure 21: Client interface - DC-Plasma power generator

This interface is invoked with a user-defined set of parameters. These parameters are programmed when the applet is placed in a web page during the process system server configuration. A simple web page at the server site would invoke the JAVA applet with the following html file:

```
<HTML>
<HEAD>
<TITLE>Front Panel HTML</TITLE>
</HEAD>
<BODY>
<APPLET CODE="DCGpanel1.class" WIDTH=458 HEIGHT=275>
    <PARAM NAME=MacId value=45>
    <PARAM NAME=MaxPower value=10000>
    <PARAM NAME=MaxVoltage value=1000>
    <PARAM NAME=MaxCurrent value=2100>
    <PARAM NAME=MaxRampTime value=1000>
```

```
    <PARAM NAME=NumDevInputs value=5>
    <PARAM NAME=NumDevOutputs value=9>
    <PARAM NAME=ServerPort value=8000>
</APPLET>
<P><BR>
<P>
</BODY>
</HTML>
```

No further configuration is necessary to invoke and operate this interface. The user only needs to have access to the Internet or an Intranet to be able to communicate with the process control system server. In addition, a web browser that supports latest JAVA technology (JDK 1.1.5) must be used.

This interface is divided in three logical sections of functions:

1) The LCD panel, twelve buttons, and a *DC ON* LED, located at the top of the interface. The buttons are labeled: *DC ON*, *FAULT RESET*, *REG*, *RUN MODE*, *A/B*, *REM ENABLE*, *MENU*, *MENU EXIT*, <, >, +, - Of these buttons, only the first two are currently enabled.

   a) The LCD panel is used to display the most important readbacks from the actual remote fieldbus device, a DC plasma power generator. These readbacks are displayed at the top row of the LCD panel. The bottom row of the panel displays the power output user command (user setpoint) and generator status messages.

2) The *START/STOP SCAN* button, the *Bus Addr:* label, the *OUT* and *IN* textboxes, and the *Server* and *Status* labels located at the middle of the interface.

   a) The *START/STOP SCAN* button is labeled *"START SCAN"* when the client is connected to the server but is not reading or writing data to the remote fieldbus device. When the user presses this button, the client interface starts scanning the remote fieldbus device for I/O (starts receiving generator readbacks and sending generator commands). After the button has been pressed, the button label changes to *"STOP SCAN"* Pressing this button again causes the client to stop scanning for I/O.

b) The *Bus Addr:* label is used to display the MACID (fieldbus address) of the device that this client interface controls. In our example, the remote generator uses fieldbus address 45.

c) The *OUT* textboxes are used to display the raw readback data (generator output data) that is received from the server.

d) The *IN* textboxes are used to display the raw command data (generator input data) that is sent to the server.

e) The *Server* label indicates the name and IP address of the server this client interface is connected to.

f) The *Status* label indicates the status of the client-server connection. This label will display *"Connected to Server"* when the client is communicating with the server, or *"Cannot connect to Server"* when the client could not connect to the server at the given server IP address and port number.

3) The utilities located at the bottom of the interface. These utilities are:

a) A real-time graphing utility

b) A remote data logging utility

c) A remote process recipe utility

Real-time graphing

When the *DISPLAY* button is pressed, the following window appears:

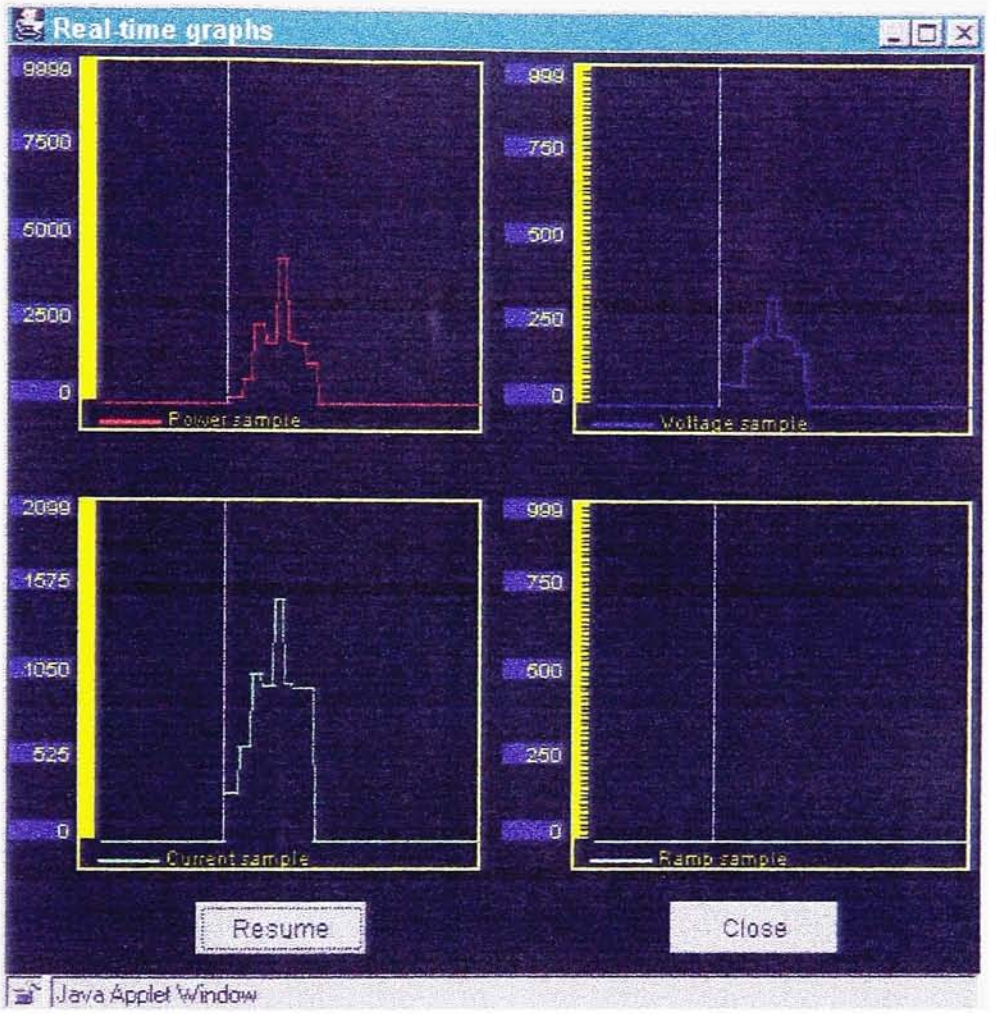

Figure 22: Client interface – real-time graphing

In this window, real-time graphs for four generator readback values are displayed. The graphs are updated simultaneously every 100ms. The above figure shows an example of readbacks for power, voltage, current and ramp time during normal operation. The *RESUME* button resumes real-time updates after the user has paused graphing. After the *RESUME* button is pressed, it is relabeled as *"PAUSE"* and will pause graphing when

pressed again. The *CLOSE* button causes the real-time graphing to stop and the window to close.

Remote data logging

This utility allows the user to obtain a log of all remote fieldbus device readbacks for a specified number of steps and step duration. When the *START* button in the main applet window is pressed, the client requests from the server to begin data logging. In this implementation, the number of data logging steps was set to 10 and the step duration to 100ms per step. Future implementations should allow the user to specify the number of steps and step duration. After the *START* button has been pressed, it is relabeled as *"STOP"* and allows the user to stop data logging at any time (possibly before the specified number of data steps have been completed). To obtain and view the data log, the *RECEIVE* button must be pressed. This causes the following window to appear:

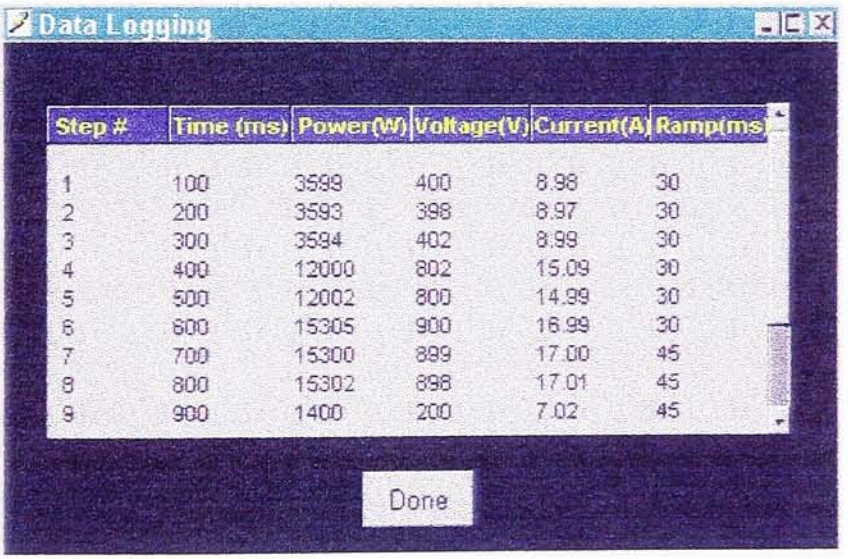

Figure 23: Client interface – data logging

The above window displays a sample data log for one generator run. For example, at 400ms, the generator output power was 12000 W, the output voltage was 802 Vdc, the output current was 15.09 Adc and the ramp time setpoint was 30 ms.

Remote process recipe

This utility allows the user to specify a list of commands that will be sent to the remote fieldbus server and will be executed in real-time. Each of these commands, called "recipe step" can be specified when the user presses the *CREATE* button and the following window appears:

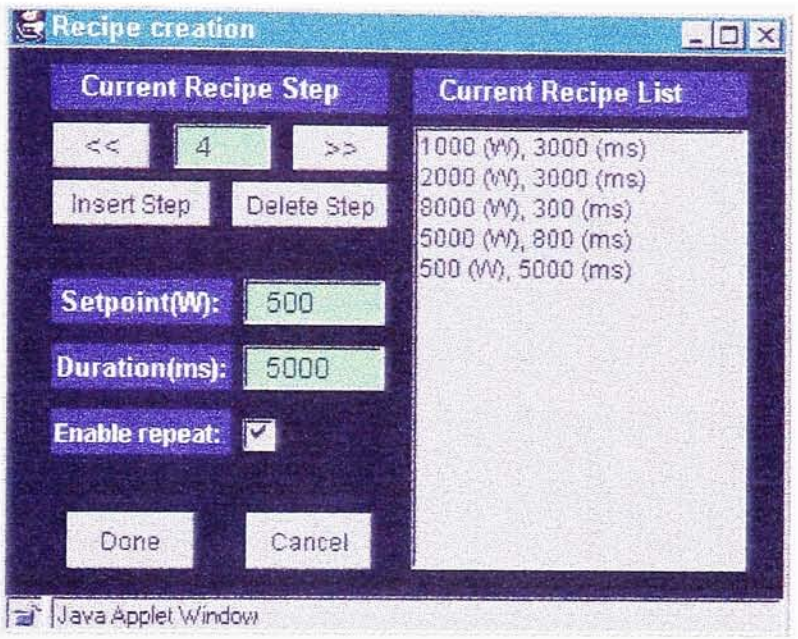

Figure 24: Client interface – remote process recipe creation

This window displays a sample recipe of five steps. Steps are numbered from 0 to 4 and contain output power setpoint (output power command) information and step duration. Once the above recipe is submitted to the server and the recipe starts running, the generator will run at 1000 W for 3 sec, 2000 W for 3 sec and so on.

To create a recipe from the beginning, the setpoint and duration must be entered in the *Setpoint* and *Duration* textboxes respectively. Then, the *Insert Step* button should be pressed to save the specified step information and the >> button must be pressed to advance to the next step. A step can be deleted by selecting the step in the *Current Recipe*

List and then pressing the *Delete Step* button. To cause the recipe to repeat infinitely during execution the user must check the *Enable repeat* checkbox. For example, this would cause the recipe in the above figure to continue running at step 0 after completing step 4.

After the recipe is constructed, and the user presses the *Done* button, the recipe is temporarily stored at the client. To transmit the recipe to the server, the *RUN* button in the main window must be pressed. This causes the client to transmit all recipe steps and to request from the server to begin running the recipe. The *RUN* button is relabeled to *STOP* and can thereafter be used to request from the server to stop running the recipe. Finally, when the *RESET* button is pressed, the client requests from the server to stop running the recipe at the current step and to continue running at step 0.